

# Événement Halloween

Bonjour et bienvenue à tous à cet événement d'Halloween ! Ce matin, nous allons tenter de créer notre potion magique d'Halloween. D'habitude, nous avons tendance à perdre la recette, mais cette fois-ci, nous l'avons ! Le seul problème, c'est que le chaudron est cassé... Quoi qu'il en soit, l'objectif de cette matinée est d'obtenir une potion bien finie !

## Avant de commencer

Pour commencer, vous allez devoir télécharger certaines choses afin de pouvoir interagir avec notre chaudron magique. Pour ce faire, il vous faut :

- [Visual Studio Code](#)
- [Node.js](#)
- [Le code de notre chaudron cassé](#)

## C'est parti !

Afin de lancer notre générateur de potion d'Halloween (en réalité c'est un site web 😊), il vous faudra ouvrir deux terminaux et aller dans le projet que vous venez de télécharger :

- Dans le premier, il faudra aller dans le dossier /API et lancer les commandes :
  - o npm install (installe toutes les bibliothèques utilisées dans le BackEnd).
  - o npm run start (lance la partie serveur et base de données du projet).
- Dans le deuxième, il faudra aller dans le dossier /FrontEnd et lancer les commandes :
  - o npm install (installe toutes les bibliothèques utilisées dans le FrontEnd).
  - o npm run dev (lance la partie client du projet).

Une fois ces terminaux lancés et les commandes exécutées, vous pouvez les minimiser et les laisser tourner en arrière-plan. Toutes les erreurs apparaîtront dans le premier terminal, c'est celui-ci que vous allez utiliser pendant cette matinée.

Vous pouvez maintenant ouvrir le navigateur internet de votre choix et y entrer  
Connexion

Il faut maintenant créer un compte et se connecter. Pour cela, rien de plus simple : vous n'avez même pas besoin d'adresse email. Il vous faut uniquement un nom d'utilisateur et un mot de passe (pas besoin de mettre quelque chose de compliqué). Si quelqu'un d'autre a déjà choisi le même nom d'utilisateur que vous, cela ne fonctionnera pas. Cela nous permet de sauvegarder votre progression si vous fermez tout accidentellement. Une fois votre utilisateur créé, vous pouvez entrer sur le site web.

## Bienvenue dans notre espace de création !

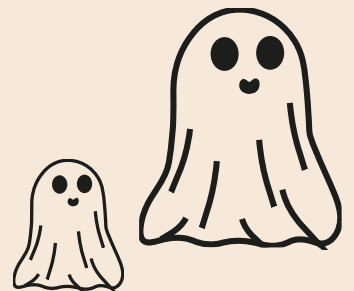
Vous pouvez enfin apercevoir notre chaudron cassé ! il n'a pas l'air comme ça, mais on a beau cliquer sur tous les boutons, rien ne fonctionne. Pour y faire des modifications, nous allons devoir ouvrir le fichier API/app.js avec Visual Studio Code.

Il va falloir dépoussiérer ce fichier ! Les citrouilles d'Halloween ont mis des trous partout dans le code ! (en vrai c'est de ma faute mais sinon vous n'auriez rien eu à faire). Chaque bloc de code que vous apercevez dans ce fichier sont des « routes » qui permettent aux informations d'aller de votre page web jusqu'à votre base de données. Par chance, le site web et la base de données sont intacts. Les premières routes sont intactes et concerne votre connexion et la création de vos identifiants, nous n'y toucherons pas.

## Route « /reset »

La première route qui a des soucis est à la ligne 90 :

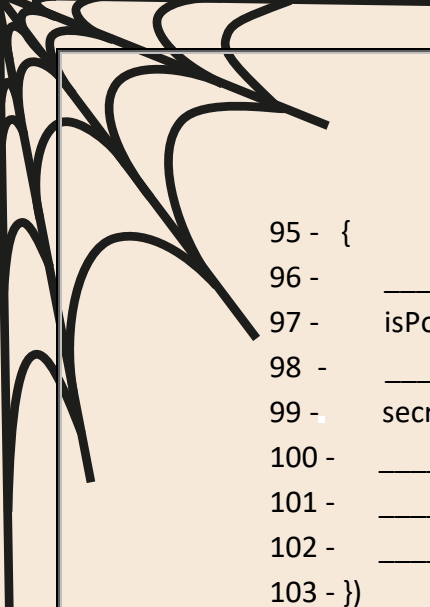
```
90 - app.patch('_____', (req, res) => {
```



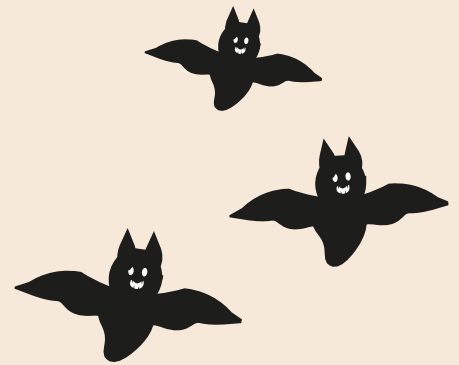
L'objectif de cette route est de tout remettre à 0, vous en aurez souvent besoin dans la matinée. Vous l'avez bien compris en ouvrant le fichier, nous sommes en train de faire de la programmation, et donc, tout est en anglais. Nous cherchons donc un nom à donner pour la route entre les guillemets. Vous pouvez prendre exemple sur les autres routes pour voir comment ça fonctionne.

Appelons-la « /reset ». Cette route est appelée à chaque fois que le bouton rouge « Réinitialiser » est appuyé.

En dessous, de la ligne 96 à 103, nous avons toutes les valeurs à remettre à 0 :



```
95 - {  
96 -   _____ : __,  
97 -   isPotionFinished : false,  
98 -   _____ : __,  
99 -   secretIngredient : "",  
100 -  _____ : __,  
101 -  _____ : __,  
102 -  _____ : __  
103 - }}
```



Tous les ingrédients sont anglais, avec tous les mots attachés et des majuscules à tous les mots sauf le premier. Par exemple pour l'ingrédient secret, nous avons « secretIngredient : "" » pour laisser un texte vide. Essayez de retrouver le nom anglais de tous les ingrédients et mettez-y le nombre 0 en face.

Petit indice : le premier ingrédient sera : « batWing, 0, » pour l'aile de chauve-souris

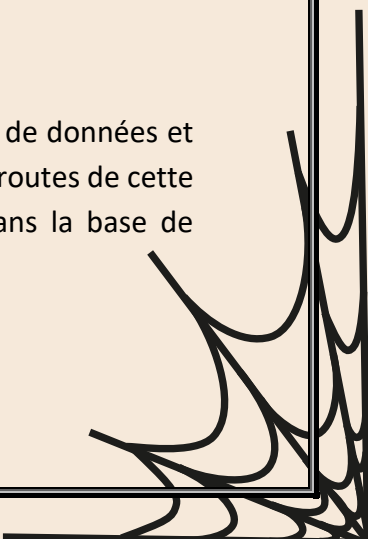
Les lignes ci-dessus permettent de réinitialiser les informations côté base de données, mais qu'en est-il du site web ? Et bien il suffit d'envoyer la même chose dans les lignes 105 à 111. Une fois les informations envoyées à la base de données et au site, notre route « /reset » est terminée.

## Route « /spiderOne »

L'objectif de cette route est de faire fonctionner le bouton +1 en dessous de la patte d'araignée afin de commencer à rajouter des éléments dans notre chaudron. Il s'agit donc de la route présente à la ligne 119 :

```
119 -   app.patch('_____', (req, res) => {
```

Comme pour la route « /reset », la route est déjà connectée à la base de données et sur le bon bouton sur le FrontEnd du site web. Cela sera le cas pour toutes les routes de cette activité. Il va nous falloir rajouter 1 au compteur de pattes d'araignées dans la base de données.



```
124 - db.collection(dbCollection).doc(req.body.username).update({
125 -     _____ : doc.data().spiderLeg + _____
126 - })
```

Sur les 3 trous à combler vous allez devoir, comme sur la route « /reset », préciser le nom de la variable que vous voulez modifier à gauche, et lui attribuer une valeur à droite. Vous avez déjà trouvé le nom de la variable à modifier dans la route « /reset » pour la partie gauche. A droite, « doc.data().spiderLeg » représente le nombre de patte d'araignée actuel dans votre chaudron. Par conséquent il faudra lui en ajouter 1 pour que notre opération soit terminée.

Une fois la valeur actualisée dans la base de données il va falloir renvoyer ce résultat au site internet. Pour se faire, nous avons la ligne 127 :

```
127 - res.status(200).json({ _____ : doc.data().spiderLeg + _____, responseType: "Success",
    errorType : null })
```

Ici, nous avons juste besoin de renvoyer la même valeur que nous venons de mettre dans la base de données mais au site web. En effet, ce format, nommé le format JSON est très souvent employé par les bases de données et les site web afin qu'ils puissent fonctionner ensemble.

## Route « /spiderFive »

Cette fois-ci nous allons nous occuper du bouton à coté ! Le +5 ! Comme vous avez pu le voir, la recette demande parfois une grosse quantité d'ingrédients. Même si cliquer frénétiquement sur le +1 est possible, nous allons voir comment il serait possible d'accélérer un peu les choses avec de la programmation.

```
134 - app.patch('_____', (req, res) =>
```

Vous l'avez compris, l'objectif de cette route sera d'ajouter les pattes d'araignées 5 par 5. Pour cela nous allons de nouveau devoir faire le lien entre la base de données et la partie FrontEnd.

```
139 - db.collection(dbCollection).doc(req.body.username).update({
140 -     _____ : doc.data().spiderLeg + _____
```



```
141 -    })
142 -    res.status(200).json({ _____: doc.data().spiderLeg + _____, responseType: "Success",
    errorType: null })
```

Après avoir fait la route « /spiderOne », vous devrez être en capacité de correctement réécrire les lignes 140 et 142, cependant, il ne faut pas oublier que les pattes d'araignées doivent s'ajouter 5 par 5 !

## Route « /spider »

Il faut savoir qu'en programmation web, les routes permettent de faire transiter (presque) toutes les informations que l'on souhaite et dans tous les sens. Pour l'instant nous n'avons fait qu'apporter des informations de notre base de données au FrontEnd mais il est également possible de récupérer des informations de ce dernier pour les envoyer en base de données. Pour ça, nous allons appeler notre route « /spider » sans nombre supplémentaire pour essayer de rajouter le nombre de pattes d'araignées que l'on souhaite.

```
149 -    app.patch('_____', (req, res) => {
```

Pour que cela puisse fonctionner, nous allons recevoir un argument du FrontEnd appelé « number » et qui sera le nombre de pattes d'araignée que nous voudrions ajouter au chaudron. Cet argument est récupéré dans l'objet « req.body ». Vous avez sûrement déjà aperçu un « req.body.username » un peu partout, il s'agit là de votre nom d'utilisateur qui est envoyé sur toutes les routes afin que vous ayez votre propre progression.

```
154 -    db.collection(dbCollection).doc(req.body.username).update({
155 -        spiderLeg: doc.data().spiderLeg + req.body._____
156 -    })
157 -    res.status(200).json({ spiderLeg: doc.data().spiderLeg + req.body._____, responseType:
    "Success", errorType: null })
```



Maintenant que vous avez compris comment fonctionne la modification en base de données et sur le site web, nous allons pouvoir accélérer 😊 .

## Routes « /batOne », « /batFive » et « /bat »

Nous avons enfin fini de nous occuper des araignées, place aux ailes de chauves-souris ! Vous pouvez déjà commencer à remplir les noms de vos nouvelles routes aux lignes 165, 180 et 195 :

```
165 - app.patch('_____', (req, res) => {  
180 - app.patch('_____', (req, res) => {  
195 - app.patch('_____', (req, res) => {
```

La première route pour ajouter 1 aile de chauve-souris, la deuxième pour en ajouter 5 et la dernière pour en ajouter autant qu'on veut.

Ensuite, nous allons nous attarder sur le contenu de ces 3 routes.



```
170 - db.collection(dbCollection).doc(req.body.username).update({  
171 -   _____: doc.data()._____ + _____  
172 - })  
173 - res.status(200).json({ _____: doc.data()._____ + _____, responseType: "Success",  
    errorType: null })  
  
185 - db.collection(dbCollection).doc(req.body.username).update({  
186 -   _____: doc.data()._____ + _____  
187 - })  
188 - res.status(200).json({ _____: doc.data()._____ + _____, responseType: "Success",  
    errorType: null })  
200 - db.collection(dbCollection).doc(req.body.username).update({  
201 -   _____: doc.data()._____ + req.body._____  
202 - })  
203 - res.status(200).json({ _____: doc.data()._____ + req.body._____, responseType:  
    "Success", errorType: null })
```

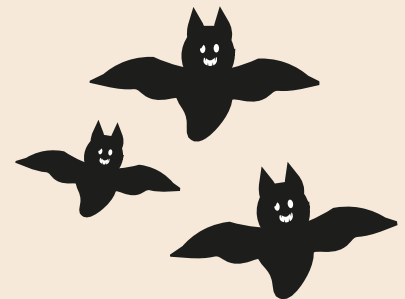
Pour remplir tout ça, vous pouvez reprendre tout ce que vous avez fait pour les pattes d'araignées. Vous pouvez voir que les 3 routes sont construites de la même manière et que seule la valeur que l'on ajoute à la base de données change par rapport aux autres.

De plus, vous pouvez remarquer que seul le nom des variables change d'un ingrédient à l'autre. Par conséquent, vous êtes enfin prêt pour réparer une grosse partie de nos routes !

## Réparons nos routes !

Vous êtes maintenant en capacité de réparer toutes les routes suivantes comme des développeurs aguerris :

- « /pumpkinOne »
- « /pumpkinFive »
- « /pumpkin »
- « /toadOne »
- « /toadFive »
- « /toad »
- « /snakeOne »
- « /snakeFive »
- « /snake »



/!\ Attention /!\ : Il y a de plus en plus de trous dans les routes 😊.

### Pour aller plus loin



Si vous êtes arrivés jusqu'ici, c'est que vous avez compris comment fonctionnent une route et une API. Avec l'aide de votre intervenant, vous allez pouvoir découvrir un peu comment fonctionne la partie Front-End de notre site web.

Il existe une dernière route nommée « /secretIngredient » qui n'est rattachée à aucun bouton sur la partie Front-End, et c'est à vous de la créer. L'objectif de cette route est de recevoir un texte avec le nom de l'ingrédient secret pour finaliser la potion. Si vous parvenez à trouver, dans tous les fichiers, quel est l'ingrédient secret, vous pourrez terminer votre potion. Vous remarquerez également qu'une fonction située tout en bas du fichier app.js permet de vérifier si votre recette est bien complète. À vos claviers ! Et bonne chance 😊