Devoir node.JS

Promotion 2018 - 2019 de la licence pro MIW. Durée 3h.

Vous réaliserez ce devoir par groupe de deux. Les premiers exercices proposés sont à faire par tout le monde. Les exercices suivants sont tous optionnels. Vous pouvez choisir de faire ceux qui vous semblent les plus abordables. Certains sont plus durs que d'autres, mais vous apporteront plus de points.

Veuillez noter dans un document texte pendant tout le devoir votre avancement sur chacun des exercices. Si un exercice ne vous a pas posé de problème, écrivez simplement OK. En revanche, si vous avez commencé un exercice mais n'avez pas réussi à le terminer complètement, que vous avez un bug, voire une piste pour le résoudre… **prenez le temps d'expliquer vos difficultés par écrit**, et comment vous pensiez éventuellement pouvoir terminer l'exercice**. Votre réflexion vaut autant que votre code** (ceci dit, essayez quand même de coder quelque chose qui fonctionne…).

A la fin du devoir, vous créerez un fichier compressé portant vos deux noms, contenant tout votre code (attention à ne pas compresser le dossier node_modules !!) et votre fichier d'explications.

Vous disposerez d'internet afin de lire des documentations, d'éplucher StackOverflow ou de regarder des vidéos de chat si vous avez décidé d'abandonner mentalement.

Essayez de coder le plus proprement possible, mettez des commentaires, indentez correctement votre code, nettoyez vos tentatives infructueuses avant de passer à l'exercice suivant… Je serai attentif à la qualité globale de votre code.

Je suppose que **personne n'aura le temps de tout faire**, choisissez donc bien vos exercices.

Bon courage !

1 - Exercices à réaliser par tout le monde

On se chauffe gentiment…

1.A : Écriture d'une promesse basique

Commencez par créer un fichier nommé *easyPromise.ts* dans le dossier *utils* de votre projet.

Vous allez y écrire une fonction nommée *letsCount* que vous exporterez. Celle-ci va compter de 0 à 3 avec un intervalle de 1 seconde. Vous utiliserez *console.log* pour affficher les chiffres dans le terminal.
-> Vous pouvez soit utiliser *setTimeout* que vous rappellerez à chaque itération, soit *setInterval*. Pour la seconde option, n'oubliez pas de l'arrêter avec un *clearInterval* après avoir fini de compter !

Modifiez ensuite votre fonction pour qu'elle retourne une promesse. Cette promesse doit se résoudre lorsque vous avez fini de compter.

Importez ensuite *letsCount* dans le fichier qui initialise l'API (*api/index.ts).* Vous appellerez votre promesse une fois votre API chargée. Lorsque celle-ci se sera résolue, affichez un message de confirmation dans le terminal.

Bonus : améliorez *letsCount* pour pouvoir prendre en paramètre le chiffre jusqu'au quel compter (*max)* et l'intervalle entre deux chiffres (*intervalDuration).*

1.B : Installer et utiliser une dépendance NPM

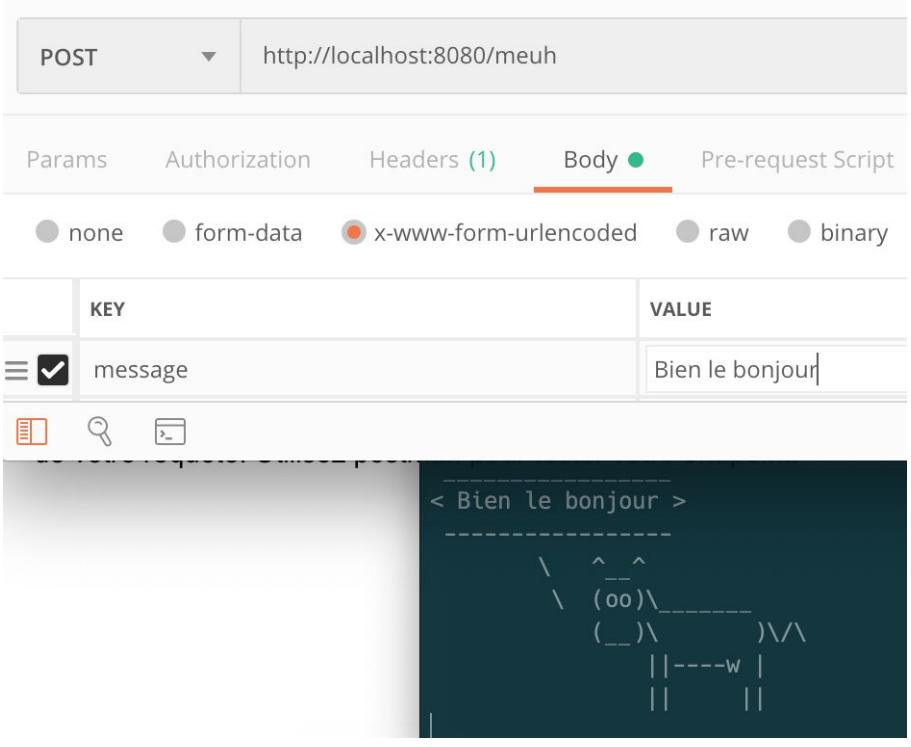
Commencez par installer la dépendance *cowsay* : https://www.npmjs.com/package/cowsay. Notez la version qui a été installée dans votre document texte.

Créez ensuite un fichier *cow.ts* dans le dossier *api/endpoints*. Vous allez y déclarer un endpoint POST sur la route */meuh*.
-> N'oubliez pas d'initialiser votre route dans le fichier *api/endpoints/index !*

Votre endpoint retournera le JSON suivant : *{ "status": "success" }*.

De plus, vous devrez importer le module précédemment installé, et faire en sorte d'afficher une vache dans votre terminal comme décrit dans la documentation : https://www.npmjs.com/package/cowsay#usage-in-the-browser

Les paroles de votre vache devront être le paramètre *message* passé dans le *body* de votre requête. Utilisez postman pour tester votre endpoint.



1.C : Les additifs c’est mal, *m’voyez*

L’une des forces de Yuka est de faire une synthèse claire et concise des additifs présents dans les produits que vous scannez. Ces additifs sont triés par dangerosité, et donnent donc une idée aux consommateurs de la qualité de ses produits. Nous allons donc reproduire ce fonctionnement.

Mettez à jour le schéma de *products* afin d’y rajouter un tableau de string. Vous nommerez ce nouveau champ *additives* et le placerez au sein de l’objet *nutrition.*

Ensuite, faites les modifications nécéssaires sur le endpoint */food/:code* afin que ce nouveau champ soit retourné dans vos appels à l’API. Vous pouvez commencer par rajouter manuellement ces valeurs dans votre collection via studio 3T avec ces valeurs d’exemple : *E621, E627, E150a, E500…*

Lorsque cela fonctionne, faites en sorte de pouvoir passer les additifs dans les paramètres body du endpoint /food/insert/:code. Trouvez un moyen élégant™ d’insérer ce tableau dans votre donnée.

2 - Exercices au choix, 1ère partie

Je crois en vous

2.A : Améliorer la gestion des additifs (difficulté moyenne)

Vous avez précédemment ajouté une gestion basique des additifs dans la question 1.C. Essayons maintenant de l'améliorer afin d'indiquer aux utilisateurs le nom complet de l'additif et son impact potentiel sur sa santé.

Créez une collection *additives* sur votre base de données et son schéma/model associés avec la structure suivante : { _id: string, name: string, impact: number, description: string }.
Pour information, *_id* sera le nom de l'additif et *impact* pourra recevoir **uniquement** les valeurs 1, 2 ou 3.

Vous allez ensuite créer un fichier nommé *additives.ts* dans le dossier *endpoints*. Comme pour vous l'avez fait pour votre bovin parlant un peu plus tôt, n'oubliez pas d'initialiser vos endpoints.
Au sein de ce fichier, ajoutez un endpoint d'insertion *(/additive/insert/:id)* et de récupération (/additive/:id) qui liront ou modifieront votre collection *additives.*

Bonus 1 (moyen): épluchez la documentation de mongoose pour vous assurer que votre schéma n'accepte que des entiers compris entre 1 et 3 pour le champ *impact*.
Attention ! : afin que la validation se fasse avec la méthode *updateOne*, vous devez ajouter le paramètre *runValidators: true* dans le 3ème paramètre de la méthode (en plus de *upsert: true)*.

Bonus 2 (difficile) : mettez à jour votre endpoint */food/:code* afin que le champ *additives* qui retourne actuellement un tableau de string retourne un tableau contenant les documents de chaque additif pioché dans la collection correspondante.

Par exemple, pour le produit suivant :

{

name: “Produit exemple”,

...,

additives: [“E621”, “E627”]

}

votre endpoint devra dorénavant retourner :

{

name: “Produit exemple”,

…,

additives: [

{ “_id”: “E621”, name: “Glutamate monosodique”, “impact”: 1, description: “blablabla” },

{ “_id”: “*E627*”, name: “Guanylate disodique”, “impact”: 2, description: “blablabla” }

]

}

C’est-à-dire qu’à la place de simples strings, vous renverrez la description complète d’un additif.

Ce bonus est assez difficile puisqu’il vous demandera de faire plusieurs (voire une seule, selon comment vous codez) requête(s) pour chaque additif et d’associer les résultats correspondants pour faire votre retour JSON. La notation pour ceux qui auront réussi / ceux qui toucheront au but sera généreuse :)

2. B : Les bons plans de tonton Yuka (moyen+)

L'idée ici est de pouvoir proposer des alternatives plus saines à certains produits afin de faciliter les choix des consommateurs. Par exemple, il serait intéressant de proposer aux utilisateurs ayant scanné du Nutella un pot de Nocciolata qui ne contient pas d'huile de palme (en plus, c'est meilleur, déso pas déso).

Commencez par rajouter un champ *recommended* à votre schéma *product* qui sera un tableau de strings. Chaque string contiendra le code barre d'un produit alternatif recommandé.

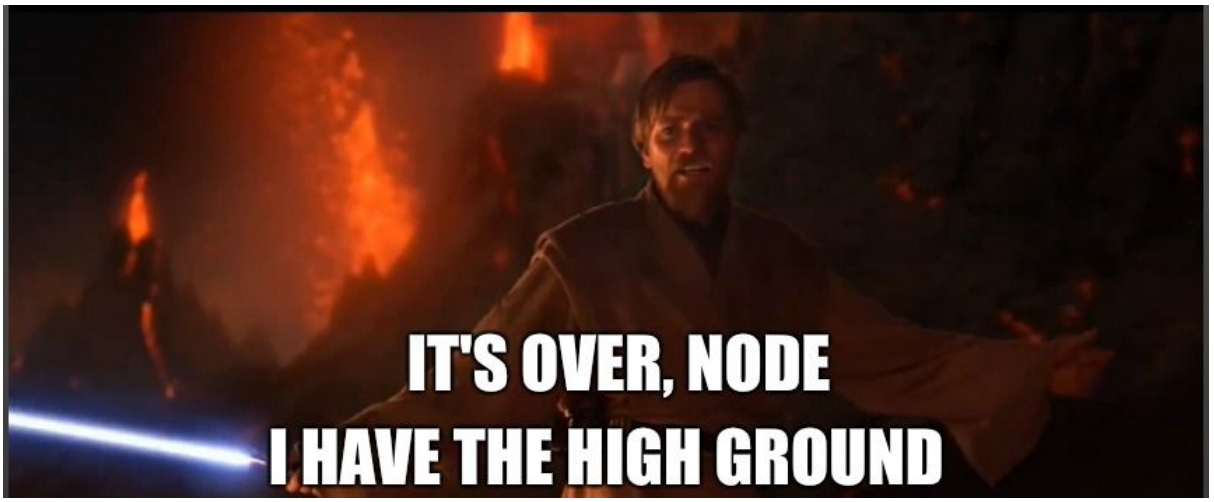
Rajoutez ensuite 2 endpoints dans le fichier *product.ts*. Le premier servira à rajouter un produit recommandé, l'autre à en enlever un. Les routes seront respectivement : */product/addRecommended/:code* et */product/removeRecommended/:code*. Le code passé en paramètre d'URL sera le code barre du produit qui recevra / perdra un produit recommandé. Le code barre du produit recommandé sera passé en paramètre body.

-> Si cela vous semble compliqué, **n'hésitez pas à relire la manière dont nous avons ajouté les noms des fichiers photos dans l'endpoint *insertPhoto*.** La logique est la même. Vous devrez probablement jeter un oeil à la documentation de mongoose pour faire la suppression d'un produit recommandé.

Si vous n'avez pas réussi à faire l'étape précédente : vous pouvez mettre à jour l'endpoint */product/insert/:code* afin de pouvoir directement insérer un tableau de produits recommandés. La logique devrait être la même que pour l'insertion des additifs dans la question 1C. J'attends toujours une solution élégante™ de votre part.

3 - Exercices aux choix, 2ème partie

Ca devient un peu plus chaud



3.A - Faire appel à son *droit de vidage (difficile)*

Pour une raison totalement obscure, votre client vous demande un endpoint facile à utiliser permettant de vider l'intégralité des photos de produits qui ont été envoyées. Cela n'a aucun sens, mais n'arrivant pas à le raisonner, vous finissez par accepter de coder cette fonctionnalité inutile.

Rajoutez un endpoint avec la méthode *OPTIONS* dans le fichier *product.ts* avec la route */product/deletePhotos* qui va supprimer l'intégralité des fichiers contenus dans le dossier *static/images*. Vous devrez pour cela lire la documentation de node, ou tout simplement StackOverflow pour trouver une manière de faire ça.

N'hésitez pas à coder des vérifications qui vous semblent intéressantes au sein de votre endpoint et d'envoyer un JSON de confirmation en rapport avec le résultat de votre suppression.

Bonus culture : si vous avez compris le nom de cet exercice, expliquez-le dans votre document texte et je vous rajouterai *peut-être* un demi point sur votre note.

3.B - *5/5, best product ever* (difficile)

Vous l'aurez peut-être compris. On souhaite rajouter un système de notation pour les produits. Celui-ci aura la forme d'une note sur 5 et fera la moyenne de l'ensemble des notes données par les utilisateurs.

Vous allez pour cela rajouter un endpoint POST, dont le nom de la route sera à votre choix. Celui-ci acceptera un paramètre *score* qui acceptera un entier allant de 1 à 5.
Cette valeur s'ajoutera dans un nouveau champ *scores* de votre collection *products* et sera un tableau d'entiers contenant toutes les notes attribuées à un produit.

Vous veillerez à effectuer les vérifications habituelles dans votre endpoint (existence du produit, renvoi d'un JSON en cas d'erreur…) et vous prendrez en compte les cas potentiellement problématiques qui peuvent vous venir en tête.

Enfin, vous allez non pas renvoyer ce tableau *scores* dans l'endpoint */product/:code* mais un seul champ *score* qui contiendra la moyenne de toutes les notes. Il existe plusieurs méthodes afin d'obtenir cette moyenne. Vous pouvez utiliser celle qui vous semble la plus simple, la plus rapide à mettre en oeuvre ou la plus optimisée.