TP n°3: Maintenance et documentation d'une application

Dans ce tp, vous aller apprendre à:

- Documenter votre code avec jsdoc
- Faire des tests unitaires basiques

1. Documenter son application

Il est important de bien documenter son application. En effet, cela à plusieurs avantages:

- Simplifier la compréhension du code
- Assurer la transmission de connaissance
- Faciliter la maintenabilité de votre application

Il existe différentes manière de documenter son application:

- Commenter les parties du programmes qui sont complexes.
- Tenir à jour une documentation fonctionnelle.
- Tenir à jour une documentation technique.
- etc...

La documentation est un exercice important mais qui peut prendre énormément de temps. Il n'est donc pas rare que cet aspect soit relégué au second plan, ce qui peut poser des problèmes (documentation manquante, non mise à jour, etc..).

Pour éviter cela, une bonne pratique est de documenter son application au fur et à mesure plutôt que de le faire en un seul coup. De nombreux outils existent pour faciliter la documentation d'une application. Dans notre cas, nous allons utiliser jsdoc.

A noter que ce que nous faisons dans ce tp correspond à de la documentation purement technique. Une application doit également avoir une documentation fonctionnelle (qui n'est pas forcement maintenue par les développeurs).

1.1 Documenter ses fonctions avec jsdoc

Jsdoc est une api permettant de générer automatiquement une documentation de votre api. Nous allons voir les bases de cet outil mais si vous voulez aller plus loin, je vous invite à regarder la documentation de jsdoc.

Principe:

jsdoc utilise des tags pour décrire les différentes parties de votre application (classes, types, function, etc...). Ici, nous allons nous contenter de décrire des fonctions.

Pour cela nous utiliserons les tags suivants:

- function
- param

- returns
- async

Pour décrire une fonction avec jsdoc, il faut ajouter un bloc de commentaire au dessus de la fonction souhaitée et d'ajouter les tags associés:

```
/**
* @function  somme
* @param  {number}  a
* @param  {number}  b
* @returns  {number} Somme de a et de b
*/
function  somme(a,b){
    return  a+b;
}
```

Et voilà, votre fonction est documentée, c'est aussi simple que ça!

Générer votre documentation

Une fois que vous avez documenté toutes vos fonction, vous pouvez générer une documentation automatiquement. Pour cela, il suffit d'installer le package npm de jsdoc globalement: `npm i jsdoc --global`

une fois cela fait, il suffit de taper la commande `jsdoc src` *Avec src le dossier contenant le code de votre application*

Votre documentation est alors disponible dans un dossier 'out'

A partir de maintenant, vous devrez fournir une jsdoc pour chaque fonction que vous allez créer lors de vos tp.

2. Tester votre application

Les tests unitaires sont une **étape essentielle** dans la création d'une application. En effet, ils possèdent de nombreux avantages:

- Assurer la qualité du code
- Gagner du temps (automatisation)
- Limiter la régression
- Améliorer la robustesse de l'application

Principe:

Le principe de base des tests unitaires (TU) consiste à écrire du code permettant de tester automatiquement une **partie précise** de votre application dans un **cas bien précis** (qu'on appelle généralement cas de test).

Exemples:

- Tester le comportement de la fonction somme dans un déroulement "classique" (a et b sont des entiers).
- Tester le comportement de la fonction somme quand a est une string et b un entier.

```
/**
* @function  somme
* @param  {number}  a
* @param  {number}  b
* @returns  {number} Somme de a et de b
*/
function  somme(a,b){
    return  a+b;
}
```

Il est très important qu'un test corresponde bien à **un** cas unique **d'une** partie précise de votre application. Dans le cas contraire, on sort du principe de test unitaire.

Ecriture des tests

Dans la suite de ce tp, vous allez écrire des tests basiques avec jest.

Installer jest: `npm install --save-dev jest`

Configurer jest: `jest --init`

Une fois cela fait, vous pouvez commencer à écrire vos tests. Pour lancer vos tests, il suffit de taper la commande `jest` à la racine de votre projet.

3 Travail à faire

Dans ce repos, vous trouverez plusieurs fichiers dont products.json, users.json et paniers.json. Ces fichiers contiennent des informations que nous utiliserons dans le cadre de ce tp.

Lire et essayer de comprendre le contenu du dossier src/__tests__

Lancer les tests unitaires

Implémenter les fonctions du fichier maths.js pour que les tests unitaires passent.

Implémenter les fonctions du fichier shop.js pour que les tests unitaires passent.

Ce que vous venez de faire correspond à du Test Driven Development. Cette pratique consiste à écrire des tests unitaires avant d'écrire sa fonction. Cela permet de bien réfléchir à l'ensemble des cas possible et de traiter toutes les erreurs potentielles.

Ecrivez les fonctions et les tests unitaires associés suivants en prenant en compte le plus de cas de tests possible: *Prenez en compte le plus de cas de tests possible et n'oubliez pas votre jsdoc !*

1. Ecrire une fonction qui retourne la liste des numéros de téléphone des utilisateurs de plus de 50 ans.

2. Ecrire une fonction qui pour chaque catégorie de produits, retourne les informations suivantes:

- Le libellé
- Le stock (si le stock d'un objet est inférieur à 10 on affiche"low", si il est entre 10 et 50 on affiche"meduim", si il est supérieur à 50, on affiche "high"

Exemple:

```
{
    "smartphone":[
        {libelle: "iPhone 9", dispo: "high" },
        {libelle: "iPhone X", dispo: "meduim" }
        ...
    ],
    "laptops":[
        {libelle: "MacBook Pro9", dispo: "high" },
        ...
    ]
}
```

3. Ecrire une fonction prenant un id de produit en paramètre et qui retourne la liste des ids des utilisateurs ayant acheté le produit. Si aucun utilisateur n'a acheté le produit, retourner le message suivant: "Ce produit n'est présent dans aucun panier".

4. Ecrire une fonction qui prend un id utilisateur en paramètre et qui retourne le prix total et le prix total "discount" de son (ou ses) panier(s). Si l'utilisateur ne possède pas de panier, retourner son adresse mail. Si le prix total dépasse 1000€, retourner également son adresse mail.

```
 [{idPanier:"toto", total:100, totalDiscounted: 90}]
 OU
  [{idPanier:"toto", total:100, totalDiscounted: 90, mail:"fakemail"}]
```

5. Ecrire une fonction qui prend un id utilisateur en paramètre:

- Si le client a entre 0 et 17: Throw une erreur car le site n'est pas ouvert aux mineurs.
- Si le client a entre 18 et 25: Retourner la liste des 10 produits sur lesquels on fait la plus grosse économie (différence prix et prix discount).
- Si le client a entre 26 et 50: Retourner la liste des produits avec une note supérieure à 4,7.
- Si le client a plus de 50 ans: Retourner la liste des produits appartenant à la catégorie "smartphone".

Si l'utilisateur est majeur et que la date d'aujourd'hui correspond à son anniversaire, retourner également son adresse mail.