Langage Javascript SN2 - Séance 4 - EXERCICES - Lille

Enoncés :

Pour ces exercices il faudra tout rendre dans un seul fichier zip nommé prenom-nom.zip. Les exercices seront regroupés dans un seul fichier html avec une balise <h1>Exercice1</h1> <h1>Exercice2</h1>etc…
Les exercices sont à envoyer à "zeynel.agzibuyuk1@mail-formateur.net"

Exercice 1 - Promesses (0.5pts):

Créez une Promise simple qui se résout après quelques secondes. Utilisez then pour afficher un message de succès.

Ensuite créez une Promise qui est rejetée après un délai. Utilisez catch pour gérer le rejet et afficher un message d'erreur.

Exercice 2 - Chargement de données asynchrone (0.5pts) :

Utilisez fetch pour effectuer une requête HTTP GET vers une API publique (par exemple:"https://jsonplaceholder.typicode.com/users") et affichez les données dans la console. Gérez les erreurs en utilisant une Promise.

Exercice 3 - Chargement multiple (1pts) :

Créez un gestionnaire de promesses pour effectuer plusieurs requêtes GET en parallèle vers différentes API :

- "https://jsonplaceholder.typicode.com/users",
- "https://jsonplaceholder.typicode.com/comments",
- "https://jsonplaceholder.typicode.com/photos",
- "https://jsonplaceholder.typicode.com/todos")

Une fois toutes les données récupérées, affichez-les.
En cas d'erreur sur un appel api, utilisez un "fallback" (api de secours, par exemple ici celle-ci : "https://jsonplaceholder.typicode.com/albums"), autrement dit appeler une autre api pour récupérer les données, même si les données sont différentes cela ne fait rien il faudra juste logger les données..

Exercice 4 - Mécanisme de retry (2 pts) :

Vous devez créer une fonction qui effectue une opération asynchrone (par exemple, une requête HTTP à l'API : “https://jsonplaceholder.typicode.com/users”) et met en place un mécanisme de réessai automatique (retry) en cas d'échec. Le but est d'améliorer la fiabilité des opérations asynchrones en permettant un nombre limité de tentatives de réessai.

Remarque : Pour cet exercice, votre premier appel HTTP se soldera forcément par un échec. À vous d’implémenter ce comportement.

Créez une fonction “effectuerOperationAsynchrone” qui prend en paramètre une URL à requêter.

Utilisez une Promesse pour effectuer la requête HTTP à l'URL fournie. Si la requête réussit, résolvez la Promesse avec les données reçues. Si la requête échoue (par exemple, renvoie une erreur 500, chose que vous devrez implémenter lors du premier appel), rejetez la Promesse.

Implémentez un mécanisme de réessai (retry) qui permettra de retenter la requête en cas d'échec. Vous pouvez spécifier le nombre maximal de tentatives de réessai et un délai entre les tentatives (généralement 3 essais et une seconde).

Si la première tentative échoue, attendez le délai spécifié, puis réessayez. Continuez ce processus jusqu'à atteindre le nombre maximal de tentatives ou jusqu'à ce que la requête réussisse.

Si la requête réussit lors d'une tentative de réessai, résolvez la Promesse avec les données reçues.

Si le nombre maximal de tentatives est atteint sans succès, rejetez la Promesse avec un message d'erreur indiquant que le nombre maximal de tentatives a été dépassé.

Exemple de retour attendu :

Lors de la première tentative, la Promesse est rejetée.

Si la première tentative échoue mais que la deuxième réussit, la Promesse est résolue avec les données de la deuxième tentative.

Si le nombre maximal de tentatives est atteint sans succès, la Promesse est rejetée avec un message d'erreur.

Exercice 5 - File d'attente de Promises (1pts) :

Implémentez une file d'attente de Promises. Les Promises sont ajoutées à la file et exécutées l'une après l'autre. Utilisez un système de priorité pour déterminer l'ordre d'exécution (notation de 1 : basse priorité à 5 : haute priorité).

Attendez une seconde avant d'exécuter toutes les promesses stockées dans la file d'attente par ordre de priorité.

Exemple de sortie :

Lorsque j'ajoute une promesse à ma file d'attente, elle n'est pas exécutée immédiatement. Au bout d'une seconde, mon code commence à dépiler la file d'attente pour exécuter toutes les requêtes en attente par ordre de priorité (1 étant une priorité basse et 5 étant une priorité haute)..
Les promises notées 5 devront donc s'exécuter avant les autres.

Exercice 6 - Gestionnaire de cache avec Promises (1pts) :

Créez une classe CacheManager qui utilise Promises pour gérer la récupération et le stockage des données en cache. Les Promises résolues devraient renvoyer les données en cache ou effectuer des requêtes en réseau si les données ne sont pas en cache.

Exercice 7 - Requête HTTP et Forms (4 pts)

Effectuez une requête HTTP à l'URL suivante : "https://jsonplaceholder.typicode.com/users/" grâce à une promesse. Pendant le chargement, affichez soit un logo de chargement soit le message "Chargement" dans votre page HTML et non dans la console, comme à votre habitude.

Créez une classe User qui respecte la structure fournie par l'API.

N'oubliez pas de créer d'autres classes si nécessaire (notamment quand un attribut est constitué d'un objet et non d'une valeur primitive).

Il faudra alors attribuer le retour de notre fonction à un tableau de User.

Une fois les données récupérées, le logo ou le texte doit disparaître et doit afficher à l'écran la liste de tous les users sous forme de liste, ici on affichera uniquement le "name" de notre user.

Lorsque je clique sur un utilisateur, cela m'ouvre une modal avec toutes les informations de mon utilisateur.

Créez ensuite un form qui permettra d'ajouter un utilisateur à notre liste, il faudra bien entendu contrôler CHAQUE champ de notre form pour éviter d'avoir du code vulnérable.

Contrôler chaque champ veut dire contrôler les caractères spéciaux, contrôler la longueur, le type, etc… Ce n'est pas juste un required.

Vous êtes libre d'utiliser n'importe quel framework HTML (Foundation, Bootstrap, etc.) pour la création de la page HTML.

Exemple de retour attendu :

Lorsque j'accède à l'index.html, j'ai un logo de chargement qui tourne, au bout de quelques instants il disparaît et apparaît devant moi une liste de noms d'utilisateurs.
Lors d'un clic sur l'utilisateur, j'ai bien accès à toutes ses informations, elles sont affichées à l'écran (soit via une modal, soit tout autre élément graphique de votre choix).
En dessous de celle-ci apparaît un formulaire me demandant de remplir tous les champs requis pour créer un utilisateur.

Lorsque je clique sur le bouton de validation du formulaire, je vois instantanément mon utilisateur s'ajouter à la liste et, quand je clique dessus, j'ai bien accès à toutes ses informations.

Exercice 8 - Quizz interactif (5 pts)

En utilisant l'URL suivante :
"https://opentdb.com/api.php?amount=20&category=20&difficulty=medium&type=multiple"

Créez un quizz interactif qui pose des questions à l'utilisateur et affiche un score à la fin.

L'application devra respecter les critères suivants :

Pour l'interface utilisateur :

Un écran d'accueil avec un bouton pour démarrer le quiz.
Une interface simple et intuitive pour afficher chaque question et ses choix de réponses.
Un bouton pour soumettre la réponse et passer à la question suivante.

Pour la logique du Quiz :

Logo de chargement tant qu'on n'a pas les données.
Chargement des questions à partir de l'API.
Affichage d'une question à la fois avec ses choix de réponses.
Possibilité pour l'utilisateur de sélectionner une réponse.
Passage à la question suivante après la soumission d'une réponse.

Pour la gestion des Scores :

Calcul du score basé sur les réponses correctes.
Affichage du score total à la fin du quiz.

Pour le feedback utilisateur :

Affichage d'une indication visuelle si la réponse est correcte ou incorrecte.
Bonus : Afficher la bonne réponse en cas de réponse incorrecte.

Déroulement :

L'utilisateur arrive sur l'écran d'accueil et clique sur le bouton pour démarrer le quiz.
Pour chaque question :
La question et les choix de réponses sont affichés.
L'utilisateur sélectionne une réponse et soumet son choix.
Une indication visuelle montre si la réponse est correcte ou non.
L'utilisateur passe à la question suivante.

Fin du Quiz : Après la dernière question :

Le score total est calculé et affiché.
Un message s'affiche affichant le score.

Exercice 9 : Création d'une Application de Prévisions Météo en Temps Réel (5 pts) :

Créez une application web simple mais fonctionnelle qui affiche les prévisions météorologiques en temps réel pour une ville spécifiée par l'utilisateur. Cette application doit effectuer des requêtes à une API météo et présenter les résultats de manière dynamique à l'utilisateur.

Vous pouvez utiliser l'URL suivante : “https://goweather.herokuapp.com/weather/{city}” en remplaçant “city” dans l'URL par votre ville, ex :
https://goweather.herokuapp.com/weather/Lille.

Interface Utilisateur Basique :

Commencez par créer une interface utilisateur simple avec HTML. Elle devrait inclure un champ de saisie pour que l'utilisateur puisse entrer le nom d'une ville, ainsi qu'un bouton pour soumettre la requête.

Requête API Météo :

Utilisez JavaScript pour écrire une fonction qui fait une requête à une API météo. La fonction doit récupérer les données météorologiques pour la ville entrée par l'utilisateur.

Affichage des Données Météo :

Affichez les informations météorologiques récupérées sur votre page web. Cela peut inclure la température actuelle, l'état du ciel (ensoleillé, nuageux, pluvieux, etc.), la vitesse du vent, etc.

Gestion des Erreurs :

Assurez-vous de gérer les cas où l'utilisateur entre une ville inexistante ou en cas d'erreur de l'API. Affichez un message approprié en cas d'erreur.

Bonus - Changement de Thème :

Ajoutez une fonctionnalité où l'interface utilisateur change de thème ou de couleur en fonction des conditions météorologiques (par exemple, des tons bleus pour un temps pluvieux, des tons jaunes pour un temps ensoleillé).