Mini Projet – Quiz multi-utilisateurs

L'objectif de ce projet est de réaliser la version multi-utilisateurs d'un jeu de quiz dont les sources vous sont fournies. Vous devrez vous appuyer sur la librairie React pour la partie cliente et NodeJs pour la partie serveur.

Principe du jeu :
Le jeu consiste, pour chaque joueur à répondre correctement aux questions affichées en choisissant une réponse parmi celles-proposées.

L'application dispose de trois pages différentes.
L'accueil qui permet aux joueurs de saisir un pseudo (unique) pour jouer. Un compteur indique le nombre de joueurs connectés et une table affiche la liste les joueurs connectés. Un bouton permet de démarrer le jeu pour l'ensemble des joueurs connectés (afficher le quiz).

Le quizz qui permet de réaliser le jeu en répondant aux questions.
Lorsqu'un joueur choisit une réponse il voit les réponses des autres joueurs pour la même question (si ceux-ci ont répondu). Lorsque tous les joueurs ont répondus à une même question, il devient impossible de la modifier et la réponse correcte est affichée. La figure ci-dessous présente les différentes possibilités :

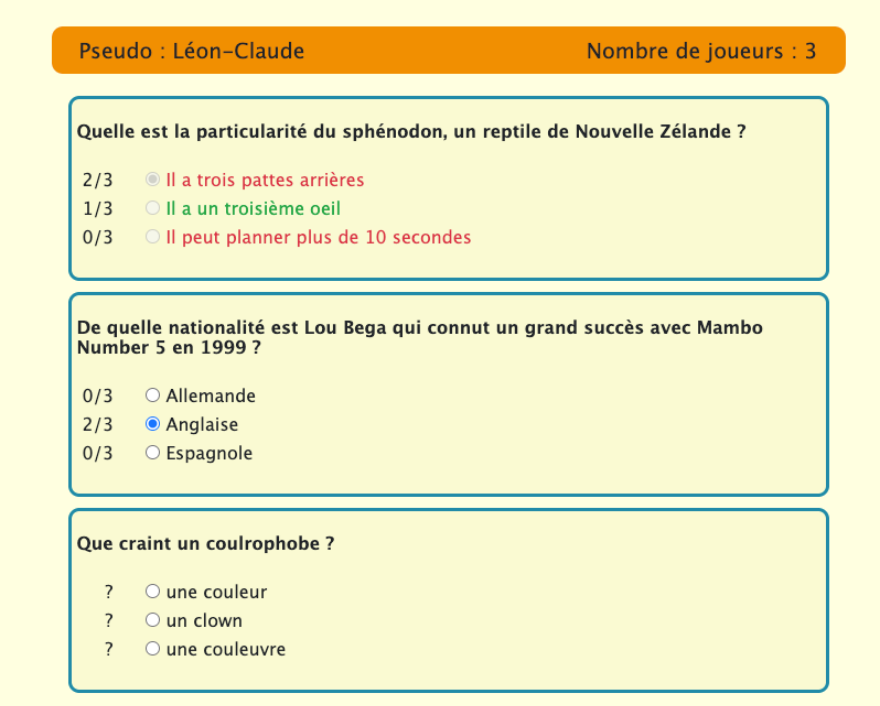

Figure 1 : En haut, question terminée par tous les joueurs., Au milieu, question répondu par le joueur mais toujours modifiable car certains joueurs n'ont pas répondu. En bas, question non répondue par le joueur.

La page de résultats
A la fin du jeu, c'est à dire quand tous les joueurs ont répondus à toutes les questions, les scores sont affichés dans la table des joueurs.

Pseudo	Score
Léon-Claude	1
Isa-Belle	2
Lou-Zer	0

Figure 2 : Table des score à la fin de la partie.

Travail attendu :
Vous devez utiliser les sources fournies et ajouter les fonctionnalités suivantes :

- Mettre en place les pages
- Créer des composants React pour la saisie du nom, la table de joueurs et les questions ainsi que les compteurs de réponses.
- Demander un pseudo (unique) au joueur
- Afficher combien d'autres joueurs sont connectés dans un compteur et quels sont leurs pseudo dans une table.
- Lancer la partie lorsqu'un des joueurs clique sur le bouton démarrer la partie.
- Afficher leurs réponses aux différentes questions pour respecter le principe du jeu.
- Calculer les scores et identifier le gagnant à la fin du jeu

Il n'est pas nécessaire de gérer les cas où un ou plusieurs joueurs s'ajoutent ou se retirent en cours de partie mais il faut le prendre en compte avant le début de la partie.

Contraintes techniques :
Le projet sera réalisé avec des technologies JavaScript et en particulier NodeJS associé aux Framework Express et Socket.IO pour la partie côté serveur. Pour le code côté client vous devez utiliser React et une organisation en grille avec Bootstrap.

Code fourni :
L'archive contient du code pour démarrer le projet. Le code est constitué de deux dossier :

- **server** : contenant l'ensemble des sources pour le serveur (dont les dépendances et des cripts de lancement sont données dans le fichier package.json). En particulier Vous trouverez un module nommé **kwiz_module** contenant plusieurs fonctionnalités pour récupérer les questions, gérer les clients et leurs réponses. Vous n'êtes pas obligé de l'utiliser mais cela peut accélérer votre travail.
- **client :** contenant l'ensemble des sources pour le client React (dont les dépendances et des scripts de lancement sont données dans le fichier package.json).

Évaluation des fonctionnalités /15 :
Voici un barème utilisé pour noter le projet :

- Demander un pseudo au joueur et l'afficher / 2
- Afficher le nombre de joueurs connectés / 1
- Afficher la liste de tous les clients connectés dans une table / 2

Lancer le jeu lorsqu'un des joueurs clique sur démarrer / 1

- Afficher le nombre de fois qu'une réponse a été choisie par les autres joueurs lorsque le joueur choisi une réponse / 4
- Afficher la solution d'une question lorsque tous les joueurs ont répondu à cette question / 2
- Empêcher la modification lorsque tous les joueurs ont répondu à une question / 1
- Afficher les scores et le gagnant dans la table des joueurs à la fin de la partie / 2

Qualité du code / 8	Excellent	ok	Not ok
-Qualité du Html/Css/JS -Qualité de l'architecture de l'application React et composants - Qualité du code serveur	- Le code est clair structure et commenté - L'architecture des données mutables et immuable et leurs utilisations dans les composants est claire. Les changements se propagent seulement ou cela est nécessaire. Bonne utilisation des fonctionnalités React (Routes, Hooks..) - La gestion des évènements va socket io client est claire et organisée. Les appels sont fait aux clients concernés.	- Le code est structuré mais pas forcément clair - L'architecture des données présentes quelques redondances ou liens distant avec des recalcul non nécessaires. Certaines fonctionnalité sont réalisées avec js et pas des approches React. - l'utilisation des socket est correcte mais tous les messages sont envoyés à tout le monde.	-Le code n'est pas clair ni commenté - L'architecture est confuse ou présente des erreurs. Beaucoup de recalcul nécessaire. Trop de js qui ne forme pas un ensemble cohérent. - Mauvaise gestion des évènements client qui forcent à l'envoie de nombreuses informations non nécessaires ou certaines manquantes.

Conseils :
Il est important de bien concevoir les échanges entre le serveur et les clients. Il vous faut donc définir un protocole de messages simples pour les évènements. Par exemple, *('nouveau_client', nom du client)* ou encore *(nom_clients_connectés, {'clients' : ['pseudo1', 'pseudo2','pseudo3']})*.

Pour identifier et stocker les données provenant des clients sur le serveur, il est pratique de créer un objet qui stocke en mémoire l'ID (unique) du client attribué par la socket. Cet ID s'obtient grâce à la méthode *var clientID = socket.id ;* Le module kwiz fournit est conçu pour fonctionner avec ce principe.

Améliorations possibles :
Ajouter des animations lors du changement du nombre de joueur + 0,5
Ajouter un gif de victoire récupéré via l'api Giphy sur la page finale + 0,5
https://developers.giphy.com/