Bonjour , je m’appel tony guéville et aujourd’hui je vais vous présenter le projet 10 de la formation d’openclassrooms d’intégrateur web

Ce projet est un peu différent des autres car cette fois ci nous allons pas coder l’intégralité du projet , openclassrooms nous fouris le code et la mission du projet et de débuger le code qu’on nous à fournis.

Pour ce faire on nous à donné plusieurs outils, dans un premier temps on nous à conseiller d’installé ReactDeveloperTools. Qui va permettre de consulter en temps réel la propagation du state/ context dans l’application et ses composants.

Montrer un peu le fonctionnement de react tool

Et en deuxieme outil on à Yarn test qui va venir exécuter les index.test.js qui va verifier si notre index.js est conforme au test associé. Donc la commande va venir tout test et nous ressortir les problème qu’il à trouvé ensuite c’est à nous de prendre connaisance du resultat du test pour voir ou notre code pose problème

Montrer le fonctionnement de yarn test

Maintenant je vais vous montrer que mon site fonctionne

Présentation du site

Je vais vous présenter létape 4 qui consister a rédiger un cahier de recette.

Avant de le réaliser on devait s’assurer que tous les bugs on été corrigé, le bute de ce cahier est de montrer une liste de bug qui était présent sur le site afin de pouvoir voir qu’ils ont été corrigé

Présentation du cahier de recette

Puis présenter 2 bug présent :

Scénariot 4 : (containers/Events)

Nous avons un problème avec le filtre dans un premier temps grace a react dev tool j’ai trouvé dans quel composant se trouver le filtre et ensuite en analysant celui-ci on peut voir qua la ligne 22 dans la const filteredEvents nous avons un problème

Voici une explication plus détaillée de la logique du deuxième bout de code :

* La ligne 22 : data?.events.filter((event) => event.type === type) filtre le tableau data.events pour n'inclure que les éléments dont la propriété event.type est égale à type.
* La ligne 23 : (currentPage - 1) \* PER\_PAGE calcule l'indice du premier élément de la page actuelle.
* currentPage \* PER\_PAGE calcule l'indice du dernier élément de la page actuelle.
* La ligne .slice((currentPage - 1) \* PER\_PAGE, currentPage \* PER\_PAGE) coupe le tableau filtré pour n'inclure que les éléments dont l'indice est compris entre les deux indices.
* Le code original filtrer le tableau events en fonction de l'indice de l'élément, tandis que la modification de ce bout de code filtre le tableau events en fonction de la propriété event.type de l'élément.

Malgré les modifications ça ne marchait toujours pas car dans le fichier

(Compenant/select) à la ligne 19 la donné de onChange était pas écrite. Une fois celle-ci écrite nos filtre fonctionnais bien en filtrant les events par catégorie

Je regarde le chronomètre :

Je vais vous présenter le scénariot 3 et 5

Sur cette parti nous avons un problème avec le slider et les bullet point qui ne sont pas clicable

Ligne 11 le useEffect était vide car la logique pour changer d’image étais au dessus alors qu’on peut l’encadré dans un useEffect et par la meme occasion faire fonctionner les bullet point

Ligne 12 Si les conditions sont remplies, ca signifie que vous avez des données valides pour afficher le carousel.

**setInterval**. Est une fonction qui sera exécutée périodiquement toutes les 5 secondes (5000 millisecondes).

Ligne 13 La fonction **setInterval** exécute met à jour l'index actif (**activeIndex**) en le retirant une unité à chaque fois, pour créer une boucle infinie du carousel. Puis arrive le return clear quand **data** ou **activeIndex** change.

1. Ensuite, il y a une fonction **handleDotClick** qui est destinée à gérer le clic sur les points de pagination du carousel. Lorsqu'un point est cliqué, cette fonction met à jour l'index actif (**activeIndex**) en fonction de la valeur du point cliqué.

Et enfin à la ligne 57 la principale différence entre le code de base et celui modifié est dans le fait que le code de base utilise un bouton radio pour une sélection exclusive, alors que le code modifié utilise une case à cocher pour permettre des sélections indépendantes.

Question : context/helpers :

export const getMonth = (date) => MONTHS[date.getMonth() + 1];

**date.getMonth() + 1** pour obtenir le nom du mois correspondant. Si le tableau **MONTHS** utilise un index basé sur 0 (comme le code de base), je dois utiliser **date.getMonth()** pour obtenir le nom du mois. La méthode utiliser dépend juste de la manière dont le tableau **MONTHS** est structuré.

Question 2 : (containers/form/index.js) de 1000 à 900)

en réduisant le délai dans la fonction **setTimeout** de 1000 ms à 900 ms dans ma fonction **mockContactApi**, j’ai accéléré la résolution de la promesse. qui permis à mes tests de s'exécuter plus rapidement et d'éviter les erreurs à cause de délais d'attente excessifs

c’est pour ça que **setTimeout** doit être choisie en fonction des besoins de mon application et de mes tests pour garantir un comportement asynchrone cohérent et éviter les erreurs de synchronisation.

la diference text unitaire et text intégration ;

En bref, les tests unitaires se concentrent sur des parties individuelles du code, tandis que les tests d'intégration se penchent sur l'interaction entre différentes parties de l'application. Les deux types de tests sont importants pour garantir la qualité globale d'une application.