Jest est un framework de test JavaScript permettant d'écrire et d'exécuter des tests unitaires de manière efficace et flexible.

LCOV est un outil qui génère des rapports de couverture de code, permettant d'évaluer la qualité et l'exhaustivité des tests en analysant quelles parties du code ont été effectivement exécutées.

Titre: Analyse du Rapport de Couverture de Tests

Bonjour à tous.

Dans ce rapport de couverture de tests généré par Jest, nous pouvons observer que 11 suites de tests ont été exécutées avec succès, couvrant un total de 57 tests. Le temps d'exécution total des tests est de 11.553 secondes.

Maintenant, concentrons-nous sur la couverture de code.

Globalement, la couverture de code est de 92.93%. Cela signifie que 92.93% de notre code source a été exécuté au moins une fois pendant les tests.

Regardons quelques points clés :

- **Couverture des instructions:** 92.93% des instructions ont été exécutées.
- **Couverture des branches:** 86.25% des branches ont été testées.
- **Couverture des fonctions:** 84.31% des fonctions ont été appelées pendant les tests.
- **Couverture des lignes:** 95.14% des lignes de code ont été exécutées.

Voyons maintenant quelques fichiers spécifiques:

- **routes.js et constants.js:** Ces fichiers ont une couverture complète à 100%. C'est un excellent résultat !
- **containers/Dashboard.js:** La couverture est de 87.32%, avec quelques lignes non couvertes (18-19, 94, 179). Il est important d'écrire des tests supplémentaires pour couvrir ces lignes.
- **containers/Login.js:** La couverture est de 100%, mais il y a une branche non couverte. Cela signifie qu'il existe une condition dans le code qui n'a pas été testée. Il faut ajouter des tests pour couvrir cette branche.
- **containers/NewBill.js:** La couverture est de 86.21%, avec plusieurs lignes non couvertes (29-36). Il est nécessaire d'écrire des tests pour couvrir ces lignes et améliorer la couverture globale.

En conclusion, ce rapport de couverture de tests nous donne une bonne vue d'ensemble de la qualité de nos tests. Nous avons une bonne couverture globale, mais il y a encore quelques zones à améliorer. Il est important de se concentrer sur

les fichiers et les lignes de code non couverts pour augmenter la fiabilité de notre application.

Merci pour votre attention.

Points clés à retenir:

- Couverture globale de 92.93%
- Fichiers avec couverture complète : routes.js, constants.js
- Fichiers avec couverture partielle : Dashboard.js, Login.js, NewBill.js
- Importance de couvrir les lignes et branches non couvertes

Note: Cette présentation est une base. Vous pouvez l'adapter en fonction de vos besoins et de l'analyse plus approfondie du rapport.

Mots clés: couverture de tests, Jest, lignes de code, branches, fonctions, fiabilité, application

1. Le code utilise des identifiants data-testid incorrects pour récupérer les valeurs des champs d'entrée de l'administrateur. (The code uses incorrect data-testid identifiers to retrieve the values of the administrator's input fields.)

L'objet user manque la propriété type, qui devrait être définie sur "Admin". (The user object is missing the type property, which should be set to "Admin".)

2. La fonction rows ne trie pas correctement les données avant de les afficher. (The rows function does not sort the data correctly before displaying it.)

Le code manquant pour trier les données par date décroissante a été ajouté. (The missing code to sort the data by descending date has been added.)

3. Le champ de téléchargement de fichier n'accepte que les images au format JPG ou JPEG. (The file upload field only accepts images in JPG or JPEG format.)

Le message d'erreur n'est pas correctement affiché pour les autres types de fichiers. (The error message is not displayed correctly for other file types.)

4. La facture affichée dans la section "En attente" a un style

incorrect. (The bill displayed in the "En attente" (Pending) section has incorrect styling.)

La couleur de fond de la facture est incorrecte, elle devrait être différente de celle des factures validées. (The background color of the bill is incorrect; it should be different from the color of validated bills.)

Ce plan de tests vérifie l'intégralité du parcours utilisateur, de la connexion à la déconnexion, en passant par la création et la consultation des notes de frais.
Il s'assure que toutes les fonctionnalités essentielles, telles que la validation des données, la gestion des fichiers et la navigation entre les différentes pages, fonctionnent correctement.
En outre, il couvre des aspects importants comme l'accessibilité, la performance et la sécurité de l'application.

Ces trois phrases résument l'objectif global du plan de tests : garantir que l'application fonctionne comme prévu du début à la fin, en couvrant tous les aspects du parcours utilisateur.

Pour une illustration plus détaillée, vous pourriez ajouter des éléments spécifiques comme :

La vérification de la logique métier : Le plan de tests s'assure que les calculs, les autorisations et les règles métier sont correctement appliqués.
La couverture des différents scénarios utilisateur : Il prend en compte les différents types d'utilisateurs (administrateurs, employés) et leurs actions spécifiques.
L'identification des points de défaillance potentiels : Le plan de tests met en évidence les zones où des problèmes pourraient survenir et propose des stratégies pour les détecter.

En résumé, ce plan de tests End-to-End offre une vision globale de la qualité de l'application et permet d'identifier les éventuelles améliorations à apporter.