Résumé des corrections d'erreurs

1. Erreurs dans les routes API

Erreur: Paramètre `request` non utilisé

Fichiers concernés:

- `src/app/api/notifications/mark-all-read/route.ts`
- `src/app/api/notifications/route.ts`

Solution:

- Préfixer le paramètre non utilisé avec un underscore: `_request` pour indiquer qu'il n'est pas utilisé dans la fonction
- Mettre à jour la documentation JSDoc en conséquence

Explication:
ESLint et TypeScript signalent les paramètres déclarés mais non utilisés dans une fonction. En préfixant le paramètre avec un underscore, nous indiquons explicitement qu'il n'est pas utilisé tout en maintenant la signature de fonction correcte pour les routes API Next.js.

2. Erreurs dans le service API

Erreur: Variable `disableErrorAlert` non utilisée

Fichier concerné:

- `src/services/api.ts`

Solution:

- Supprimer la variable de la destructuration des options

Explication:
La variable était présente dans la signature mais n'était jamais utilisée dans le corps de la fonction. La meilleure approche est de la supprimer complètement si elle n'est pas nécessaire, plutôt que de la préfixer d'un underscore.

3. Erreurs dans le hook useApi

Erreur: Module non trouvé

Fichier concerné:

- `src/hooks/useApi.ts`

Solution:

- Remplacer l'import utilisant l'alias `@/services/api` par un chemin relatif `../services/api`

Explication:
L'alias `@/` n'était pas correctement configuré dans le projet, causant l'erreur de module non trouvé. Utiliser un chemin relatif est une solution fiable tant que la configuration des alias n'est pas mise à jour.

Erreur: Utilisation de `any`

Solution:

- Remplacer `any` par des types plus spécifiques:
  - `transform?: (data: any) => T` -> `transform?: (data: unknown) => T`
  - `dependencies?: any[]` -> `dependencies?: unknown[]`
  - `apiCache.set(key, {...})` -> Typé correctement avec `CacheEntry<unknown>`

Explication:
Le type `unknown` est plus sûr que `any` car il force à vérifier le type avant d'effectuer des opérations sur la valeur, tout en permettant une flexibilité similaire.

Erreur: Problèmes de dépendances React Hook

Solutions:

1. **JSON.stringify de fetchOptions**:
   - Créer une variable mémorisée `fetchOptionsString = JSON.stringify(fetchOptions)`
   - Utiliser cette variable dans les dépendances des hooks
2. **Spread operator dans useEffect**:
   - Créer une variable explicite pour les dépendances: `const depsArray = [...dependencies]`
   - Ajouter un commentaire pour désactiver l'avertissement ESLint
3. **Parsing des options**:
   - Utiliser `JSON.parse(fetchOptionsString)` pour recréer l'objet fetchOptions dans execute

Explication:
React exige que toutes les dépendances d'un hook soient listées explicitement. Les objets complexes comme `fetchOptions` changent de référence à chaque rendu, ce qui cause des problèmes de boucles infinies. La sérialisation/désérialisation permet de contourner ce problème en ne déclenchant la mise à jour que lorsque le contenu change réellement.

Bonnes pratiques mises en œuvre

1. **Gestion des paramètres non utilisés**:

- Utilisation du préfixe underscore pour les paramètres requis mais non utilisés
- Suppression des variables inutilisées

2. **Typage strict**:
   - Remplacement de `any` par `unknown` pour une meilleure sécurité
   - Définition d'interfaces précises pour les types complexes
3. **Optimisation des hooks React**:
   - Minimisation des re-rendus inutiles avec la mémorisation des dépendances
   - Gestion correcte des tableaux et objets dans les dépendances
4. **Documentation**:
   - Maintien de la documentation JSDoc cohérente avec l'implémentation
   - Ajout de commentaires explicatifs pour les parties complexes du code

Ces corrections contribuent à un code plus propre, plus sûr et plus facile à maintenir.