UNIVERSITÉ
CAEN
NORMANDIE

Université de Caen Normandie

Programmation d'applications client

Rapport

TodoLists

Élèves :
Yahya QACH 22109246
Ahmed MORABET 22107739

18 novembre 2024

Chargé de TP :
FrancoisRIOULT

Bonus réalisés :
Barre de progression (ProgressBar)
Interface utilisateur agréable

Table des matières

1 Introduction

Dans le cadre de nos travaux pratiques, nous avons développé, en binôme, une application de gestion de tâches en React. Ce projet vise à concevoir un outil moderne permettant aux utilisateurs de gérer leurs tâches quotidiennes de manière intuitive et efficace. En intégrant des technologies récentes comme React Native et GraphQL, l'application reflète les standards actuels en matière de développement d'applications web et mobiles.

L'application offre à l'utilisateur les fonctionnalités suivantes :

— Création d'un compte ou connexion à un compte existant via un système d'authentification sécurisé ;
— Gestion complète des tâches : ajout, modification, suppression ;
— Interaction avancée avec la liste des tâches, incluant un compteur, des filtres pour afficher les tâches résolues ou en cours, et une barre de progression ;
— Possibilité d'ajouter des images à chaque tâche et d'exporter les listes de tâches dans des formats variés.

Le projet repose sur l'API GraphQL disponible à l'URL `http://graphql.unicaen.fr:4000`, qui assure la gestion des données côté serveur. Grâce à cette API, nous avons pu implémenter des fonctionnalités comme la gestion des utilisateurs, la création et manipulation des listes de tâches, et le suivi de l'état des tâches.

L'architecture de l'application est organisée en plusieurs modules :

— Les composants React, conçus pour être modulaires et réutilisables ;
— Le système de navigation, qui distingue les utilisateurs connectés et non connectés tout en offrant une expérience utilisateur fluide ;
— Les écrans principaux, qui forment l'interface utilisateur de l'application ;
— Les styles, rassemblés dans un fichier centralisé pour assurer la cohérence visuelle.

Dans ce rapport, nous détaillerons les différentes étapes du développement de l'application :

— La structure des composants et leur organisation ;
— L'intégration et l'utilisation de l'API GraphQL pour les interactions serveur ;
— La gestion du contexte global et des états utilisateurs ;
— Les choix d'interface et les fonctionnalités bonus implémentées.

Enfin, le rapport se conclura par une synthèse des apprentissages réalisés, les défis rencontrés, et les perspectives d'amélioration envisagées pour cette application.

2 Le composant `TodoItem`

Cette partie du projet concerne la gestion et l'affichage des éléments individuels d'une liste de tâches. Elle est implémentée via le composant `TodoItem`, qui gère chaque tâche en tant qu'objet avec des propriétés spécifiques telles que son état (`done`) et son contenu. Voici une description des principales fonctionnalités du composant :

2.1 Affichage d'une tâche

Le composant `TodoItem` représente chaque tâche avec :
- Un `Switch` qui permet de marquer une tâche comme terminée ou en cours, en modifiant dynamiquement l'état `done` via une interaction utilisateur ;
- Un texte qui affiche le contenu de la tâche avec une ligne barrée si elle est marquée comme terminée (`textDecorationLine`) ;
- Une icône de corbeille permettant de supprimer une tâche grâce à l'appel de la fonction `deleteTodo` passée en props.

2.2 Mécanisme de mise à jour d'état

Le composant utilise le hook `useState` pour gérer l'état local (`done`) de la tâche. Une fois que l'utilisateur modifie cet état via le `Switch`, la mise à jour est également transmise au parent grâce à la fonction `changeItem` :
- Cela garantit une synchronisation entre l'état local du composant et celui de la liste globale ;
- La fonction `useEffect` est utilisée pour surveiller les modifications des props et mettre à jour l'état local en conséquence.

3 Le composant `TodoList`

Ce composant gère l'ensemble des tâches et agit comme conteneur principal pour afficher, ajouter, filtrer et modifier les tâches. Voici une description détaillée de ses fonctionnalités :

3.1 Affichage et gestion des tâches

Le composant utilise `FlatList` pour afficher la liste des tâches, permettant un rendu optimisé des éléments :
- Chaque élément de la liste est rendu via le composant `TodoItem`, en passant les props nécessaires (`item`, `changeItem`, `deleteTodo`) ;
- Un compteur (`progress`) et une barre de progression permettent de visualiser le nombre de tâches terminées par rapport au total.

3.2 Ajout et suppression de tâches

L'utilisateur peut :
- Ajouter une nouvelle tâche grâce à un champ de texte (`TextInput`) et un bouton d'action ;
- Supprimer une tâche via l'icône de corbeille, ce qui met à jour dynamiquement la liste et le compteur.

3.3 Filtres et interactions avancées

Le composant offre plusieurs options pour filtrer ou manipuler les tâches :
— Afficher toutes les tâches, seulement celles en cours, ou celles terminées ;
— Marquer toutes les tâches comme faites (`Check All`) ou réinitialiser leur état (`Check None`) avec des mises à jour synchronisées côté serveur.

4 Le composant `TodoLists`

Cette partie permet de gérer plusieurs listes de tâches. Elle agit comme un tableau de bord où l'utilisateur peut visualiser, créer et supprimer des listes de tâches.

4.1 Gestion des listes

— Les listes existantes sont affichées grâce à un `FlatList`, où chaque élément est un lien interactif vers les détails de la liste ;
— L'utilisateur peut ajouter une nouvelle liste via un champ de texte ou supprimer une liste existante avec une icône de corbeille.

4.2 Navigation entre les écrans

Chaque liste est un point d'entrée vers un écran dédié (`Details`), ce qui est géré grâce à la navigation React. Cela permet une expérience utilisateur fluide et une organisation logique de l'application.

4.3 Résumé

Les composants `TodoItem`, `TodoList`, et `TodoLists` constituent l'essence de l'application, en offrant une gestion flexible et intuitive des tâches et des listes. Leur conception modulaire et réutilisable garantit une expérience utilisateur optimisée et un code maintenable.

5 L'intégration de l'API GraphQL

Cette section décrit la manière dont l'application utilise l'API GraphQL pour gérer les tâches et les listes de tâches. Les fonctionnalités implémentées incluent l'authentification des utilisateurs, la gestion des tâches, et la gestion des listes de tâches.

5.1 L'API de base

Le projet utilise l'URL de l'API suivante :

`http://graphql.unicaen.fr:4000`

Chaque interaction avec l'API suit le standard GraphQL, où des requêtes et mutations spécifiques sont définies pour gérer les données.

5.2 Gestion des utilisateurs

Deux mutations principales permettent de gérer l'authentification des utilisateurs :
- `signIn(username, password)` : Authentifie un utilisateur avec son nom d'utilisateur et son mot de passe, et retourne un token.
- `signUp(username, password)` : Permet à un nouvel utilisateur de s'enregistrer dans le système.

Ces fonctionnalités sont implémentées en JavaScript avec la bibliothèque `node-fetch`. En cas d'erreur, un message approprié est retourné pour informer l'utilisateur.

5.3 Gestion des tâches (`Todo`)

L'application fournit plusieurs fonctionnalités pour interagir avec les tâches, telles que :
- **Créer une tâche** : Mutation `createTodo(content, todoListId, token)` permet d'ajouter une nouvelle tâche dans une liste existante.
- **Récupérer des tâches** : La requête `getTodos(todoListId, token)` récupère toutes les tâches associées à une liste donnée.
- **Mettre à jour une tâche** : Mutation `updateTodo(todoId, done, token)` met à jour l'état d'une tâche (terminée ou non).
- **Supprimer une tâche** : Mutation `deleteTodo(id, token)` supprime une tâche spécifique en fonction de son `id`.

5.4 Gestion des listes de tâches (`TodoList`)

En plus des tâches, l'application prend en charge les listes de tâches avec les fonctionnalités suivantes :
- **Créer une liste de tâches** : Mutation `createTodoList(username, title, token)` permet de créer une nouvelle liste de tâches associée à un utilisateur.
- **Récupérer des listes de tâches** : La requête `getTodoLists(username, token)` retourne toutes les listes appartenant à un utilisateur donné.
- **Supprimer une liste de tâches** : Mutation `deleteTodoList(id, token)` supprime une liste de tâches existante.

5.5 Implémentation technique

Les appels à l'API sont réalisés en utilisant la méthode HTTP POST, avec des headers spécifiant le type de contenu (`Content-Type: application/json`) et l'autorisation (`authorization: Bearer token`). Les données sont formatées en JSON, incluant la requête ou mutation GraphQL et ses variables.

5.6 Traitement des erreurs

Chaque réponse est vérifiée pour détecter les erreurs éventuelles. Si des erreurs sont présentes, elles sont remontées et affichées à l'utilisateur ou consignées dans la console pour faciliter le débogage.

5.7 Résumé

L'intégration de l'API GraphQL permet à l'application de gérer efficacement les utilisateurs, les tâches et les listes de tâches. L'utilisation de mutations et requêtes spécifiques rend l'application flexible et extensible, tout en maintenant une expérience utilisateur fluide.

6 La gestion de la navigation

L'application utilise le système de navigation de React Navigation pour gérer les différents écrans et interactions utilisateur. Ce système permet une expérience fluide et intuitive pour naviguer entre les fonctionnalités de l'application.

6.1 Structure de la navigation

Le fichier `Navigation.js` définit la logique de navigation principale. Deux types de navigateurs sont utilisés :

- **Navigateur à onglets (Tab Navigator)** : Permet de basculer entre les différents écrans principaux, comme la connexion, les listes de tâches, et la déconnexion.
- **Navigateur empilé (Stack Navigator)** : Utilisé pour naviguer entre des écrans liés, par exemple, pour afficher les détails d'une liste de tâches à partir de l'écran des listes.

6.2 Gestion des utilisateurs connectés et non connectés

La navigation s'adapte dynamiquement en fonction de l'état de l'utilisateur :

- Si le `token` est `null` (non connecté), seuls les écrans de connexion (`SignIn`) et d'inscription (`SignUp`) sont disponibles.
- Si un `token` valide est présent (utilisateur connecté), les écrans principaux (`Home`), `TodoLists`, et `SignOut` sont affichés.

6.3 Navigateur empilé : `NavigationTodo`

Ce sous-navigateur gère les écrans relatifs aux listes de tâches :

- `TodoListsScreen` : Affiche toutes les listes de tâches.
- `TodoListDetailsScreen` : Affiche les détails d'une liste spécifique.

Cette structure facilite une navigation claire et hiérarchique entre les listes et leurs détails.

6.4 Personnalisation de la barre de navigation

Des styles spécifiques (`tabBarStyles`) sont appliqués à la barre de navigation pour offrir une interface utilisateur agréable et cohérente. Ces styles incluent :

— La couleur de fond de la barre ;
— Les icônes et étiquettes pour identifier clairement chaque onglet.

6.5 Résumé

Le système de navigation de l'application est conçu pour être réactif et intuitif. En distinguant les utilisateurs connectés et non connectés, et en utilisant une combinaison de navigateurs à onglets et empilés, l'application garantit une expérience fluide et bien structurée.

7 Les écrans de l'application

L'application est composée de plusieurs écrans principaux, chacun servant une fonction spécifique pour gérer l'interaction utilisateur. Ces écrans sont implémentés sous forme de composants React Native dans le dossier `Screens`. Voici une description détaillée de chaque écran.

7.1 Écran `HomeScreen`

Cet écran agit comme le tableau de bord principal après la connexion de l'utilisateur.

— Affiche un message de bienvenue personnalisé contenant le nom d'utilisateur.
— Fournit un bouton pour accéder à la liste des tâches (`TodoLists`).
— Simplifie l'expérience utilisateur en servant de point d'entrée vers les fonctionnalités principales.

7.2 Écran `SignInScreen`

Permet à un utilisateur de se connecter à l'application.

— Contient deux champs (`TextInput`) pour le nom d'utilisateur et le mot de passe.
— Valide les informations d'identification via l'API GraphQL (`signIn`).
— En cas de succès, génère un token pour l'utilisateur et le redirige vers l'écran `HomeScreen`.
— Gère les erreurs en affichant des messages appropriés (par exemple, échec de l'authentification).

7.3 Écran `SignOutScreen`

Permet à un utilisateur de se déconnecter de l'application.

— Réinitialise le `token` et le nom d'utilisateur dans le contexte global.
— Redirige l'utilisateur vers l'écran `SignInScreen`.

7.4 Écran `SignUpScreen`

Permet à un utilisateur de créer un nouveau compte.
— Contient des champs similaires à `SignInScreen` pour le nom d'utilisateur et le mot de passe.
— Envoie les informations au serveur via l'API GraphQL (`signUp`).
— Gère les erreurs et redirige l'utilisateur vers `HomeScreen` après la création réussie du compte.

7.5 Écran `TodoListsScreen`

Affiche toutes les listes de tâches disponibles pour l'utilisateur connecté.
— Utilise le composant `TodoLists` pour afficher et gérer les listes.
— Permet d'accéder aux détails d'une liste spécifique.

7.6 Écran `TodoListDetailsScreen`

Affiche les détails d'une liste de tâches sélectionnée.
— Utilise le composant `TodoList` pour afficher et gérer les tâches associées à la liste.
— Récupère l'ID de la liste via les paramètres de navigation (`route.params`).

7.7 Résumé

Les écrans constituent la structure visuelle et interactive de l'application. Ils permettent à l'utilisateur de naviguer de manière intuitive entre les différentes fonctionnalités. L'utilisation de composants réutilisables (`TodoLists`, `TodoList`) garantit une modularité et une maintenabilité optimales.

8 Conclusion

Le développement de cette application de gestion de tâches en React a été une expérience riche en apprentissages, tant sur le plan technique que collaboratif. En travaillant en binôme, nous avons pu renforcer nos compétences en développement front-end et back-end, tout en apprenant à mieux gérer les différentes étapes d'un projet, de la conception initiale à la réalisation finale.

Grâce à l'utilisation de technologies modernes comme React Native et GraphQL, nous avons conçu une application fonctionnelle, intuitive et modulaire, capable de répondre aux besoins des utilisateurs en matière de gestion de tâches personnelles. L'intégration de fonctionnalités avancées, telles qu'une barre de progression et l'ajout d'images pour les tâches, a permis d'améliorer l'expérience utilisateur tout en respectant les bonnes pratiques de développement.

Nous avons également été confrontés à plusieurs défis, notamment l'implémentation de l'API GraphQL et la gestion des états complexes dans l'application. Ces difficultés nous ont poussés à approfondir notre compréhension des outils et à adopter des solutions robustes et efficaces.

Ce projet nous a permis de mettre en pratique des concepts théoriques appris en cours, comme la gestion des contextes globaux, la modularité des composants, et l'intégration d'API. Il nous a également permis de développer des compétences transversales, telles que la collaboration, la communication, et la résolution de problèmes.

Pour aller plus loin, plusieurs perspectives d'amélioration peuvent être envisagées :

— Ajouter des fonctionnalités supplémentaires, comme des notifications pour les échéances des tâches ou une meilleure personnalisation de l'interface utilisateur ;
— Optimiser les performances de l'application, notamment pour la gestion des grandes listes de tâches ;
— Explorer le déploiement de l'application sur des plateformes mobiles et web pour toucher un public plus large.

En conclusion, ce projet nous a permis d'allier théorie et pratique, tout en nous préparant à relever de futurs défis dans le domaine du développement logiciel.