Application de Gestion de Tâches en TypeScript

Objectif: Construire une application de gestion de tâches permettant à l'utilisateur de créer, afficher, modifier, supprimer des tâches, et de les catégoriser.

Fonctionnalités:

- **Création et Affichage des Tâches:** Permettre à l'utilisateur d'ajouter des tâches avec des détails tels que le titre, la description, la date d'échéance, et la priorité (Faible, Moyenne, Haute).
- **Catégorisation des Tâches:** Offrir la possibilité de créer des catégories personnalisées pour organiser les tâches.
- **Validation des données :** Permettre la validation des données et remonter les erreurs si les informations ne sont pas valides
- **Modification et Suppression:** Permettre la modification et la suppression de tâches existantes.
- **Filtrage et Tri:** Ajouter des fonctionnalités pour filtrer et trier les tâches par date d'échéance, priorité, ou catégorie.
- **Persistance des Données:** Utiliser localStorage pour sauvegarder et récupérer les tâches du navigateur.

Technologies:

- TypeScript pour toute la logique de l'application.
- HTML/CSS pour la structure et le style de l'application.
- localStorage pour la persistance des données.

Instructions Détaillées

1. Préparation de l'Environnement

- Initialiser un nouveau projet TypeScript.
- Préparer les fichiers index.html et style.css pour l'interface utilisateur.

2. Structure de l'Application

- **Modèles de Données:**
  - Définir des interfaces pour une tâche (Task) et une catégorie (Category).
- **Logique de Gestion des Tâches:**
  - Créer des classes pour la gestion des tâches (TaskManager) et des catégories (CategoryManager).

Application de Gestion de Tâches en TypeScript

3. Fonctionnalités de l'Application

- **Création de Tâches:**
  - Formulaire pour saisir les détails de la tâche.
- **Affichage des Tâches:**
  - Afficher les tâches dans une liste ou une grille, avec toutes les informations pertinentes.
- **Validation des données:**
  - Système de validation des données.
- **Modification et Suppression:**
  - Boutons ou liens pour modifier et supprimer chaque tâche.
- **Catégorisation:**
  - Possibilité d'ajouter des catégories et d'assigner des tâches à ces catégories.
- **Filtrage et Tri:**
  - Options pour filtrer et trier les tâches.

4. Persistance des Données

- Utiliser localStorage pour sauvegarder et charger les tâches et catégories.

Bonus supplémentaire :

- **Recherche**: Implémenter une fonction de recherche pour trouver des tâches par mots-clés.

Ce que j'attends de vous :

Dans le cadre de cet exercice, vous êtes encouragés à incorporer et à démontrer votre compréhension des concepts clés de TypeScript suivants. L'objectif est de vous familiariser avec les fonctionnalités puissantes de TypeScript, améliorant ainsi la qualité, la maintenabilité et la robustesse de votre code. Voici les concepts que vous devez explorer :

- **Types de Base et Types Avancés :** Utilisez des types primitifs (comme string, number, boolean) et des types avancés (comme les enum, tuple, et les types personnalisés) pour définir de manière précise la structure de vos données de tâche.

- **Interfaces et Types d'Union :** Définissez des interfaces pour les objets de tâche et utilisez des types d'union pour gérer différentes formes de données de manière flexible.

- **Génériques :** Utilisez des fonctions génériques pour créer des composants réutilisables qui peuvent travailler avec différents types tout en conservant leur sécurité de type.

- **Décorateurs :** Expérimentez avec les décorateurs pour ajouter des métadonnées ou des comportements supplémentaires à vos classes et méthodes de manière déclarative.

- **Espaces de Noms et Modules :** Organisez votre code en modules ou en espaces de noms pour améliorer sa structure et sa maintenabilité.

- **Asserations de Type :** Utilisez des asserations de type pour informer le compilateur du type spécifique d'une variable dans les situations où vous en savez plus que TypeScript.

- **Contrôle de Flux Basé sur les Types :** Appliquez des vérifications de type et des gardes de type pour gérer le flux d'exécution de manière sûre en fonction des types de données manipulés.

- **Opérateurs Avancés :** Exploitez les opérateurs avancés comme le type nullish coalescing (??) et l'opérateur de chaînage optionnel (?.) pour gérer les valeurs null ou undefined de manière concise.

- **Fonctions Fléchées et this :** Utilisez des fonctions fléchées pour gérer les contextes this dans vos callbacks ou fonctions d'événement, assurant que this se réfère à l'objet attendu.

- **Promesses et Async/Await :** Gérez les opérations asynchrones avec les promesses et le syntagme async/await pour une manipulation plus claire et plus lisible des processus asynchrones.

Rappel sur le localstorage :

Le **localStorage** est une fonctionnalité de stockage web qui permet aux applications web de stocker des données sous forme de paires clé-valeur directement dans le navigateur de l'utilisateur, sans date d'expiration. Cela signifie que les données stockées dans le **localStorage** sont sauvegardées entre les sessions de navigation : elles restent disponibles après la fermeture et la réouverture du navigateur.

Mega Bonus :

Synchronisation avec une Fausse API Externe

Objectif : Développer une fonctionnalité de synchronisation qui permet aux utilisateurs de sauvegarder et de récupérer leurs tâches via une fausse API externe, simulant l'interaction avec un backend.

- Utilisez **json-server** ou tout autre outil similaire pour créer une fausse API REST. Si vous ne pouvez pas installer de nouveaux outils, créez simplement des fonctions mock en TypeScript qui simulent le comportement d'une API en utilisant des délais (avec setTimeout) pour imiter les temps de réponse réseau.
- La fausse API devrait avoir des endpoints pour obtenir, ajouter, modifier, et supprimer des tâches.

Pour créer un projet Typescript avec node :
npm init -y
npm i --save-dev typescript @types/node
npx tsc —init (permettra de générer tsconfig.json)