Problème : Gestion d'une plateforme de cours en ligne (E-learning)

I. Contexte

Une startup souhaite développer une plateforme de cours en ligne en *TypeScript*. Les étudiants doivent modéliser et implémenter une partie du système en respectant les contraintes suivantes.

II. Énoncé

1. *Définition des types de base*
   a) Créez un type `Role` qui peut valoir `"student" | "teacher" | "admin"`.
   b) Définissez une interface `User` avec :
      - `id: number`
      - `name: string`
      - `role: Role`
      - `email?: string` (optionnelle).
2. *Programmation orientée objet (POO)*
   a) Créez une classe abstraite `Course` avec :
      - une propriété `title: string`
      - une méthode abstraite `getDescription(): string`.
   b) Implémentez deux classes :
      - `VideoCourse` (ajoute `duration: number`)
      - `LiveCourse` (ajoute `date: Date`).
3. *Interfaces et Génériques*
   a) Créez une interface générique `Repository<T>` avec les méthodes :
      - `add(item: T): void`
      - `remove(id: number): void`
      - `getAll(): T[]`
   b) Implémentez une classe `UserRepository` qui gère une liste de `User`.
4. *Types avancés*
   a) Créez un type `ApiResponse<T>` qui peut contenir :
      - `data?: T`
      - `error?: string`
   b) Définissez une fonction `fetchUser(id: number): ApiResponse<User>` qui renvoie soit un utilisateur, soit une erreur.
5. *Modules et imports/exports*

   Supposez que vous organisiez le code en fichiers :

   - `models.ts` → contient les types et interfaces (`User`, `Role`, `ApiResponse<T>`).
   - `courses.ts` → contient les classes `Course`, `VideoCourse`, `LiveCourse`.
   - `repositories.ts` → contient `Repository<T>` et `UserRepository`.
   - `main.ts` → contient la simulation suivante :

a) Créez deux utilisateurs (un étudiant, un professeur).
b) Ajoutez-les dans le `UserRepository`.
c) Créez un `VideoCourse` et un `LiveCourse`.
d) Affichez la description des cours et la liste des utilisateurs.

III. Questions

1. Donnez le code TypeScript correspondant à chaque étape (types, classes, interfaces, fonctions).
2. Expliquez la différence entre *interface* et *type* dans l'usage que vous avez fait.
3. Justifiez pourquoi `Course` doit être *abstraite*.
4. Montrez la différence entre un *export nommé* et un *export par défaut* dans votre projet.
5. Proposez une amélioration possible de ce système (par ex. gestion générique de `CourseRepository`).