Phase 1 : Mise en place des fichiers et structure de base

1.1 Créer la structure des fichiers

- Créer un répertoire principal pour le projet, par exemple `qcm_app/`.
- À l'intérieur, créer les fichiers suivants :
  - `main.py` : Le fichier principal qui contiendra la logique du programme.
  - `utilisateurs.json` : Fichier pour stocker les informations des utilisateurs.
  - `questions.json` : Fichier pour stocker les questions et options de QCM.
  - `resultats.csv` ou `resultats.txt` (facultatif) : Pour exporter les résultats si vous l'implémentez.

1.2 Initialiser les fichiers JSON (ou CSV)

- Créer un fichier `utilisateurs.json` avec la structure de base pour les utilisateurs.

```
json
Copier le code
{
  "utilisateurs": []
}
```

- Créer un fichier `questions.json` avec des questions et options de QCM.

```
json
Copier le code
{
  "questions": [
    {
      "question": "Quel est le langage de programmation utilisé dans
ce projet ?",
      "options": ["Python", "Java", "C++", "JavaScript"],
      "bonne_reponse": "Python"
    },
    {
      "question": "Quel est le résultat de 2 + 2 ?",
      "options": ["3", "4", "5", "6"],
      "bonne_reponse": "4"
    }
  ]
}
```

Phase 2 : Développement des fonctionnalités de base

2.1 Gestion des utilisateurs

- Implémenter la gestion des utilisateurs dans le fichier `main.py`.
  - Créer une fonction `load_users()` pour charger les utilisateurs depuis `utilisateurs.json`.
  - Créer une fonction `save_users()` pour enregistrer les modifications dans `utilisateurs.json`.
  - Implémenter une logique pour demander à l'utilisateur son nom/ID.

- Vérifier si l'utilisateur existe dans le fichier :
  - Si oui, afficher son historique de QCM.
  - Si non, permettre à l'utilisateur de créer un profil.

2.2 Gestion des questions

- Implémenter la fonction `load_questions()` pour charger les questions et options depuis `questions.json`.
- Afficher les questions une par une, avec les options, et permettre à l'utilisateur de répondre en choisissant une option.

2.3 Gestion des scores

- Calculer et afficher le score de l'utilisateur après qu'il ait répondu à toutes les questions du QCM.
- Enregistrer le score et la date dans l'historique de l'utilisateur.

2.4 Feedback à l'utilisateur

- Après chaque question, afficher un message indiquant si la réponse est correcte ou incorrecte.
- En cas de mauvaise réponse, afficher la bonne réponse.

Phase 3 : Améliorations et ajout de fonctionnalités avancées

3.1 Implémentation du chronomètre

- Ajouter un chronomètre pour limiter le temps de réponse par question (par exemple, 30 secondes par question).
  - Utiliser la bibliothèque `time` pour suivre le temps de réponse de l'utilisateur.

3.2 Catégories de questions

- Ajouter la possibilité de choisir une catégorie avant de commencer le test (par exemple, Python, Réseaux, Algorithmes).
  - Organiser les questions dans `questions.json` par catégorie :

```
json
Copier le code
{
  "Python": [
    {
      "question": "Quel est le résultat de 2 + 2 ?",
      "options": ["3", "4", "5", "6"],
      "bonne_reponse": "4"
    }
  ],
  "Réseaux": [
    {
      "question": "Quel protocole est utilisé pour le web ?",
```

```
            "options": ["HTTP", "FTP", "SMTP", "POP3"],
            "bonne_reponse": "HTTP"
        }
    ]
}
```

3.3 Exportation des résultats

- Ajouter une fonctionnalité pour exporter les résultats dans un fichier texte ou CSV.
    - Utiliser la bibliothèque `csv` pour exporter les résultats au format CSV.
    - Exemple :

```
python
Copier le code
import csv
with open('resultats.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Utilisateur', 'Date', 'Score'])
    writer.writerow([username, date, score])
```

3.4 Amélioration de l'interface

- Ajouter des messages d'accueil plus détaillés pour guider l'utilisateur.
- Ajouter des options pour démarrer un nouveau QCM, afficher l'historique, ou quitter l'application.

Phase 4 : Tests et Débogage

4.1 Tests unitaires

- Tester chaque fonction individuellement :
    - Tester `load_users()` et `save_users()` pour vérifier la gestion correcte des fichiers JSON.
    - Tester le calcul du score et l'enregistrement dans l'historique de l'utilisateur.
    - Tester le chronomètre pour s'assurer qu'il fonctionne comme prévu.

4.2 Test de l'ensemble du programme

- Effectuer des tests sur l'application entière :
    - Créer plusieurs utilisateurs, passer plusieurs QCM, et vérifier l'historique des scores.
    - Tester l'affichage du feedback et des résultats.
    - Vérifier le bon fonctionnement du chronomètre et de l'exportation des résultats.

Phase 5 : Finalisation et Documentation

5.1 Documentation du projet

- Rédiger un fichier README pour expliquer comment utiliser l'application.
    - Exemple de sections à inclure :
        - **Installation** : Comment configurer l'application sur une machine locale.
        - **Fonctionnalités** : Description des fonctionnalités principales.
        - **Exemples** : Exemples d'exécution et résultats possibles.

5.2 Nettoyage du code

- Refactoriser le code pour améliorer la lisibilité (noms de variables et fonctions claires, suppression des redondances).
- Ajouter des commentaires dans le code pour expliquer les parties complexes.

5.3 Derniers tests

- Tester une dernière fois l'application pour s'assurer que tout fonctionne comme prévu avant de soumettre le projet.

Résumé des étapes principales :

1. **Phase 1** : Préparation des fichiers et structure de base.
2. **Phase 2** : Implémentation des fonctionnalités principales (gestion des utilisateurs, des questions, des scores).
3. **Phase 3** : Ajout de fonctionnalités avancées (chronomètre, catégories, exportation des résultats).
4. **Phase 4** : Tests, débogage et validation du programme.
5. **Phase 5** : Finalisation (documentation et nettoyage du code).

Ce plan devrait vous guider dans le développement du projet de manière progressive. Vous pouvez ajuster le plan en fonction de vos besoins ou des contraintes de temps.

Répartition des tâches pour une équipe de 3 personnes

---

1. Personne 1 (Rahma) : Gestion des utilisateurs & Interface principale

Tâches principales :

- **Gestion des utilisateurs** :
    - Implémenter la création et la gestion des profils d’utilisateurs (vérification si l’utilisateur existe déjà, création d’un nouveau profil si nécessaire).

    - Implémenter la fonction `load_users()` pour charger les utilisateurs depuis le fichier JSON.
    - Implémenter la fonction `save_users()` pour enregistrer les utilisateurs et l'historique des scores.
    - Afficher l'historique des QCM, scores et dates des tests pour chaque utilisateur.
- **Interface utilisateur** :
    - Gérer l'affichage des informations de bienvenue et les instructions de l'application dans la console.
    - Permettre à l'utilisateur de choisir ses actions dans le menu (par exemple : démarrer un QCM, afficher l'historique, quitter).
    - Implémenter le flux principal du programme : chargement du profil utilisateur, affichage du QCM, gestion de la session.

Livrables :

- Code pour la gestion des utilisateurs et l'historique (en JSON).
- Code pour l'interface en console (choix des options, affichage des informations).

---

2. Personne 2 (Jasmine): Gestion des questions, réponses & Evaluation

Tâches principales :

- **Gestion des questions et réponses** :
    - Créer la fonction `load_questions()` pour charger les questions et les options depuis `questions.json`.
    - Gérer l'affichage des questions dans la console et permettre à l'utilisateur de sélectionner une réponse.
    - Vérifier les réponses de l'utilisateur et calculer le score.
    - Afficher un feedback pour chaque question après que l'utilisateur ait répondu (réponse correcte ou incorrecte, afficher la bonne réponse en cas de mauvaise réponse).
- **Évaluation et feedback** :
    - Implémenter la logique pour calculer le score final de l'utilisateur après le test.
    - Enregistrer le score et la date de passation du test dans le profil de l'utilisateur (à l'aide de `save_users()`).

Livrables :

- Code pour gérer les questions et les réponses.
- Calcul du score, affichage du feedback et mise à jour de l'historique de l'utilisateur.

---

3. Personne 3 (Saly): Fonctionnalités avancées & Exportation des résultats

Tâches principales :

- **Fonctionnalités avancées** :

    - Implémenter un chronomètre pour limiter le temps de réponse par question (par exemple, 30 secondes par question).
    - Implémenter la gestion des catégories de questions (par exemple : Python, Réseaux, Algorithmes).
        - Organiser les questions en différentes catégories dans `questions.json`.
        - Permettre à l'utilisateur de choisir une catégorie avant de commencer le QCM.
- **Exportation des résultats** :
    - Ajouter la fonctionnalité pour exporter les résultats des QCM dans un fichier CSV ou texte.
    - Créer une fonction pour enregistrer les scores des utilisateurs dans un fichier `resultats.csv` ou `resultats.txt`.

Livrables :

- Code pour le chronomètre et la gestion des catégories.
- Code pour l'exportation des résultats (CSV ou texte).

Collaboration entre les membres de l'équipe

Voici comment vous pouvez collaborer efficacement :

- **Communication constante** : Utilisez un outil de communication comme Slack ou Discord pour discuter et résoudre rapidement les problèmes.
- **Partage de code** : Utilisez un gestionnaire de version comme Git (via GitHub ou GitLab) pour gérer les versions du code et éviter les conflits. Chaque membre peut travailler sur une branche dédiée et fusionner les changements après révision.
- **Revues de code** : Faites des revues de code régulièrement pour vous assurer que le code est propre, fonctionne correctement et respecte les conventions du projet.

Résumé des responsabilités :

1. **Personne 1** : Gestion des utilisateurs et interface principale (chargement, création, historique).
2. **Personne 2** : Gestion des questions, réponses et évaluation (affichage, vérification, score).
3. **Personne 3** : Fonctionnalités avancées (chronomètre, catégories, exportation des résultats).