RAPPORT TECHNIQUE

Application de QCM Informatique

Année Universitaire 2024/2025

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Informatique

3ème année ingénieur - ing 3

Réalisé par :

Nom et Prénom	Groupe
IMANSOURA Ramy	03
IZEM Mohammed Amine	02
HAMMAMTI Mohammed Fares Yacine	02
HADDAD Chahinez	03

RÉSUMÉ

L'application intègre plusieurs fonctionnalités avancées telles que :

— La gestion du temps avec un chronomètre par question

— Le suivi des scores et l'historique des parties

— L'export des résultats en format CSV

— Une interface utilisateur intuitive

Les choix techniques ont été guidés par des objectifs de performance, de maintenabilité et d'extensibilité.

SOMMAIRE

RÉSUMÉ 1

SOMMAIRE 2

1 Introduction 4

1.1 Objectifs du Projet 4

1.2 Méthodologie 4

2 Choix Techniques et Justifications 4

2.1 Architecture du Code 4

2.2 Stockage des Données 4

2.3 Gestion du Temps 5

2.4 Interface Console 5

3 Défis Rencontrés 5

3.1 Gestion du Temps 5

3.2 Persistance des Données 5

3.3 Gestion des Sessions Utilisateur 6

4 Solutions Implémentées 6

4.1 Système de Score 6

4.2 Export des Résultats 6

4.3 Gestion des Erreurs 6

5 Améliorations Possibles 7

5.1 Techniques 7

5.2 Fonctionnelles 7

6 Métriques du Projet **7**

6.1 Performance 7

6.2 Fiabilité 7

7 Conclusion **8**

1 Introduction

Le projet QCM s'inscrit dans une démarche de modernisation des outils pédagogiques. Dans un contexte où l'évaluation continue des connaissances devient primordiale, notre application propose une solution moderne et efficace.

1.1 Objectifs du Projet

- Créer une plateforme interactive d'évaluation
- Assurer un suivi précis des performances
- Permettre une utilisation flexible et intuitive
- Garantir la fiabilité des données

1.2 Méthodologie

- Analyse des besoins
- Conception orientée objet
- Développement itératif
- Tests et validation

2 Choix Techniques et Justifications

2.1 Architecture du Code

- **Programmation Orientée Objet** choisie pour :
  - Meilleure organisation du code et encapsulation
  - Facilité de maintenance et d'extension
  - Séparation claire des responsabilités
  - Réutilisabilité des composants
- Classes principales :
  - `Timer` : Isolation de la logique temporelle
  - `QCMApp` : Centralisation de la logique métier

2.2 Stockage des Données

- **Format JSON** sélectionné pour :
  - Performance : Pas besoin de SGBD pour un petit volume de données

- Simplicité : Lecture/écriture native en Python
- Portabilité : Format universel, facile à migrer
- Debugging : Format lisible par l'humain
- Flexibilité : Structure adaptable sans schéma fixe

2.3 Gestion du Temps

- **Threading** implémenté pour :
  - Non-blocage : L'utilisateur peut répondre pendant le décompte
  - Précision : Mesure exacte du temps écoulé
  - Économie de ressources : Thread daemon qui se termine automatiquement
  - Expérience utilisateur : Retour visuel en temps réel

2.4 Interface Console

- **Interface ligne de commande** retenue pour :
  - Rapidité de développement
  - Compatibilité universelle
  - Performances optimales
  - Focus sur la fonctionnalité plutôt que l'apparence
  - Facilité de test et de débogage

3 Défis Rencontrés

3.1 Gestion du Temps

- **Problème** : Synchronisation entre le chronomètre et les entrées utilisateur
- **Impact** : Risque de conflit entre le timer et la saisie de réponse
- **Solution** :

```
def _run_timer(self):
    while self.running and self.time_left > 0:
        time.sleep(1)
        self.time_left -= 1
```

3.2 Persistance des Données

- **Problème** : Gestion des fichiers JSON inexistants
- **Solution** :

```
def initialize_files(self):
    Path("data").mkdir(exist_ok=True)
```

```
if not Path("data/users.json").exists():
    with open("data/users.json", "w", encoding="utf-8") as f:
        json.dump({}, f)
```

3.3 Gestion des Sessions Utilisateur

— **Problème** : Maintien de la session et historique
— **Solution** : Structure de données utilisateur avec historique complet

4 Solutions Implémentées

4.1 Système de Score

— Calcul en temps réel
— Sauvegarde automatique
— Format standardisé :

```
result_entry = {
    "date": datetime.datetime.now().strftime("%Y-%m-%d %H:%M"),
    "score": result["score"],
    "total": result["total"],
    "time_taken": result["time_taken"],
    "category": result["category"]
}
```

4.2 Export des Résultats

— **Format CSV** pour la portabilité
— Structure claire avec en-têtes
— Implémentation :

```
def export_results(self, username: str, filename: str):
    with open(filename, 'w', newline='', encoding='utf-8') as f:
        writer = csv.writer(f)
        writer.writerow(['Date', 'Catégorie', 'Score', 'Temps total'])
```

4.3 Gestion des Erreurs

— Validation des entrées utilisateur
— Gestion des timeouts

— Sauvegarde automatique
— Création des fichiers manquants

5 Améliorations Possibles

5.1 Techniques

— Interface graphique (GUI)
— Base de données relationnelle
— API REST pour mode multi-joueur
— Cryptage des données utilisateur

5.2 Fonctionnelles

— Mode compétition
— Système de classement
— Questions à choix multiples
— Support média (images, son)

6 Métriques du Projet

6.1 Performance

— Temps de réponse ¡ 100ms
— Utilisation mémoire ¡ 50MB
— Fichiers JSON ¡ 1MB

6.2 Fiabilité

— Sauvegarde automatique
— Validation des données
— Gestion des erreurs robuste

7 Conclusion

Les choix techniques ont privilégié la simplicité et la fiabilité, tout en gardant la porte ouverte aux évolutions futures.