



QuizAI - Documentation du Projet

Plateforme d'Apprentissage Universitaire Intelligente

Date de création : Octobre 2025

Version : 1.0

Langage : Python 3.11+

Framework : Streamlit

IA : Multi-provider (Groq, DeepSeek, xAI, OpenAI, Together)



Table des Matières

1. [Vue d'ensemble](#)
2. [Architecture du Projet](#)
3. [Fonctionnalités Principales](#)
4. [Composants Techniques](#)
5. [Fonctions Clés](#)
6. [Configuration et Déploiement](#)
7. [Guide d'Utilisation](#)








Vue d'ensemble

Objectif

QuizAI est une plateforme éducative intelligente conçue pour les étudiants, enseignants et chercheurs universitaires français. Elle utilise l'intelligence artificielle pour transformer des documents académiques en ressources d'apprentissage interactives.

Problèmes Résolus

-  **Génération automatique de quiz** à partir de cours PDF/DOCX
-  **Résumés intelligents structurés** avec mind maps, timelines, glossaires
-  **Analyse de performance** avec recommandations personnalisées

-  **Flashcards automatiques** pour révision efficace
-  **Support multi-providers IA** pour flexibilité et coûts optimisés

Technologies Utilisées

- **Frontend:** Streamlit (interface web interactive)
- **Backend:** Python 3.11+
- **IA:** OpenAI SDK (compatible multi-providers)
- **Document Processing:** PyPDF2, python-docx
- **Visualisation:** Plotly
- **Configuration:** python-dotenv

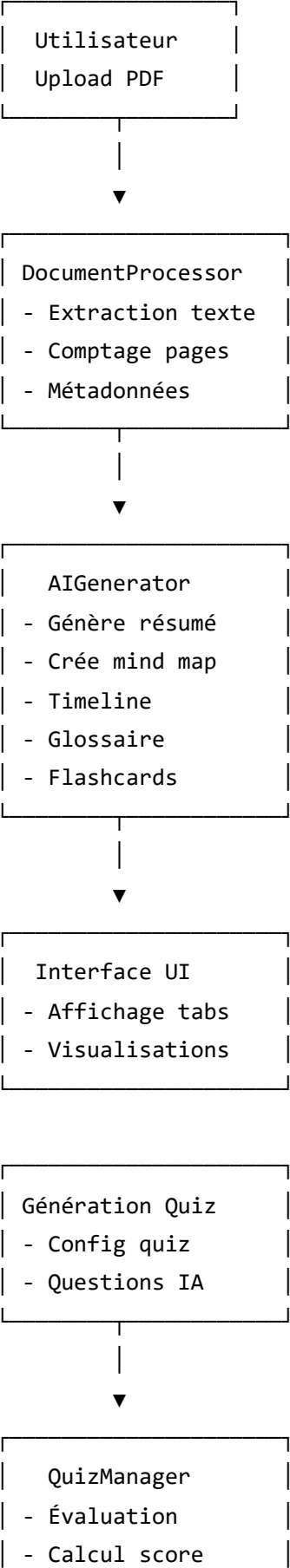


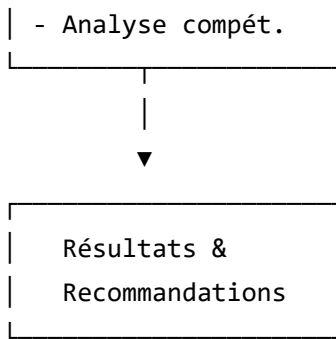
Architecture du Projet

Structure des Fichiers

```
python-INDIA/
|
├─ app.py                # Point d'entrée principal
├─ config.py             # Configuration centralisée
├─ requirements.txt      # Dépendances Python
├─ .env                  # Variables d'environnement (API keys)
|
├─ assets/
|   └─ styles.css        # Styles CSS personnalisés
|
├─ pages/                # Pages Streamlit multi-page
|   └─ 1_📄_Upload_Documents.py # Upload et analyse de documents
|   └─ 2_📝_Generate_Quiz.py    # Génération et passage de quiz
|   └─ 3_📊_Results.py          # Résultats et statistiques
|   └─ 4_💡_Recommendations.py # Recommandations IA
|
└─ utils/                # Modules utilitaires
    └─ __init__.py
    └─ ai_generator.py    # Génération de contenu IA
    └─ database.py        # Gestion de données (futur)
    └─ document_processor.py # Traitement de documents
    └─ quiz_manager.py    # Gestion et évaluation de quiz
```

Flux de Données





🌟 Fonctionnalités Principales

1. 📁 Upload et Analyse de Documents

Formats supportés: PDF, DOCX, DOC







Taille max: 10 MB par fichier

Fonctionnalités d'analyse:

- **Extraction de texte** avec comptage précis des pages
- **Mind Map visuel** - Relations entre concepts
- **Timeline chronologique** - Pour documents historiques
- **Glossaire automatique** - Termes techniques avec définitions
- **Flashcards** - Questions/réponses pour révision
- **Analyse de document:**
 - Type détecté (cours, article, rapport, thèse, manuel)
 - Niveau de difficulté (débutant, intermédiaire, avancé)
 - Mots-clés principaux
 - Concepts connexes suggérés
 - Temps de lecture estimé
 - Statistiques (pages, mots, concepts uniques)

Interface:

Tabs organisés pour navigation facile

	Analyse	# Vue d'ensemble, statistiques, mots-clés
	Résumé	# Résumé général + résumés par section
	Mind Map	# Diagramme de concepts
	Timeline	# Événements chronologiques
	Glossaire	# Termes techniques
	Flashcards	# Cartes de révision





2. Génération de Quiz Intelligente

Configuration de Quiz:



- **Types de quiz:**
 - QCM Examen
 - Questions de cours
 - Exercices d'application
 - Questions de synthèse
 - Préparation TD/TP
 - Révision finale
- **Niveaux de difficulté:**
 - Niveau TD (facile)
 - Niveau Partiel (moyen)
 - Niveau Examen Final (difficile)
- **Modes d'évaluation:**
 - Mode Contrôle Continu
 - Mode Examen Final
 - Mode Rattrapage
 - Mode Auto-évaluation
- **Barèmes:**
 - Binaire (0 ou max points)
 - Points négatifs (pénalités)
 - Partiel (points partiels)

Options avancées:

- ☒ Temps limite configurable (auto ou manuel)
- ☒ Mélange aléatoire des questions






-  Mélange aléatoire des réponses
-  Affichage optionnel des explications
-  Navigation rapide entre questions
-  Sauvegarde automatique des réponses

Interface de passage:

- Barre de progression visuelle
- Timer en temps réel
- Compteur de questions répondues
- Navigation précédente/suivante
- Sidebar avec accès rapide à toutes les questions
- Indicateurs visuels ( répondu /  non répondu)

3. Résultats et Analyse

Vue d'ensemble:

- **Statistiques globales:**
 - Quiz complétés
 - Score moyen (/20)
 - Taux de réussite (%)
 - Temps total passé
- **Graphiques interactifs:**
 - Évolution des scores dans le temps
 - Répartition des mentions (pie chart)
 - Comparaison entre quiz
- **Mentions:**
 - Excellent (≥ 16) 
 - Très Bien (14-16) 
 - Bien (12-14) 
 - Assez Bien (10-12) 
 - Insuffisant (< 10) 

Détails par Quiz:

- Score détaillé et pourcentage
- Temps écoulé
- Nombre de réponses correctes
- **Analyse par compétence:**

- Compréhension
- Application
- Analyse
- Mémorisation
- Points faibles identifiés
- Actions suggérées (télécharger corrigé, revoir stats, recommencer)

Analyse par Compétence:

- Graphique radar des performances
- Score moyen par compétence
- Progression visuelle (barres)
- Recommandations ciblées:
 - ⚠️ Compétence à travailler en priorité (<50%)
 - 📖 Nécessite plus de pratique (50-70%)
 - ✅ Bonne maîtrise (>70%)

4. 💡 Recommandations Personnalisées

Génération automatique:

Basé sur l'analyse des résultats de quiz, l'IA génère:

1. Points à revoir:

- Chapitres/sections spécifiques
- Raisons détaillées
- Actions suggérées

2. Exercices recommandés:

- Exercices ciblés par compétence
- Niveau adapté

3. Ressources supplémentaires:

- Vidéos (🎥)
- Livres (📖)
- Articles (📄)
- Cours en ligne (🎓)
- Description et liens

4. Stratégies d'apprentissage:

- Techniques de révision
- Méthodes d'étude

- Conseils pratiques

5. Planning de révision:

- Plan hebdomadaire structuré
- Actions concrètes
- Checkboxes interactives

Projection de Progression:

- Graphique de tendance avec régression linéaire
- Projection sur 5 quiz futurs
- Message motivant selon la tendance
- Comparaison avec seuil de réussite

Composants Techniques

1. config.py - Configuration Centralisée

Rôle: Gestion centralisée de tous les paramètres de l'application

```
class Config:
    # AI Provider Configuration
    AI_PROVIDER = os.getenv("AI_PROVIDER", "deepseek")

    # API Keys pour multi-providers
    DEEPSEEK_API_KEY = os.getenv("DEEPSEEK_API_KEY")
    GROQ_API_KEY = os.getenv("GROQ_API_KEY")
    XAI_API_KEY = os.getenv("XAI_API_KEY")
    OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
    TOGETHER_API_KEY = os.getenv("TOGETHER_API_KEY")

    # Modèles par provider
    MODELS = {
        "deepseek": "deepseek-chat",
        "groq": "deepseek-r1-distill-llama-70b",
        "xai": "grok-beta",
        "openai": "gpt-4o-mini",
        "together": "meta-llama/Meta-Llama-3.1-70B-Instruct-Turbo"
    }
```


Points clés:

- Chargement automatique depuis `.env` avec `python-dotenv`
- Support de 6 fournisseurs IA différents
- Configuration flexible des modèles
- Constantes pour disciplines, niveaux, types de quiz

2. `utils/ai_generator.py` - Génération de Contenu IA



Rôle: Interface unifiée pour tous les fournisseurs IA



Architecture Multi-Provider:

```
class AIGenerator:
    def __init__(self):
        self.provider = Config.AI_PROVIDER
        self.model = Config.MODELS.get(self.provider)
        self.client = self._initialize_client()

    def _initialize_client(self):
        """Initialise le client selon le provider"""
        if self.provider == "deepseek":
            return openai.OpenAI(
                api_key=Config.DEEPSEEK_API_KEY,
                base_url="https://api.deepseek.com"
            )
        elif self.provider == "groq":
            return openai.OpenAI(
                api_key=Config.GROQ_API_KEY,
                base_url="https://api.groq.com/openai/v1"
            )
        elif self.provider == "xai":
            return openai.OpenAI(
                api_key=Config.XAI_API_KEY,
                base_url="https://api.x.ai/v1"
            )
        # ... autres providers
```

Avantages:

-  Interface OpenAI SDK unifiée
-  Changement de provider en 1 ligne de config

-  Gestion automatique des erreurs
-  Extraction intelligente de réponses

3. `utils/document_processor.py` - Traitement de Documents

Rôle: Extraction et traitement de texte depuis PDF/DOCX

Fonctions principales:





```
@staticmethod
def extract_text_from_pdf(file) -> tuple[str, int]:
    """
    Extrait le texte d'un PDF et retourne (texte, nombre_pages)

    Returns:
        tuple[str, int]: (texte_extraite, nombre_de_pages_réelles)
    """
    pdf_reader = PyPDF2.PdfReader(file)
    text = ""
    for page in pdf_reader.pages:
        text += page.extract_text() + "\n"
    page_count = len(pdf_reader.pages)
    return text, page_count
```

```
@staticmethod
def process_document(file) -> Optional[Dict[str, Any]]:
    """
    Traite un document et retourne métadonnées complètes

    Returns:
        {
            "name": str,
            "type": str,
            "text": str,
            "word_count": int,
            "char_count": int,
            "page_count": int # Pages réelles, pas estimation!
        }
    """
```

Points importants:

-  Comptage de pages **réel** (pas d'estimation)
-  Gestion d'erreurs robuste
-  Support DOCX avec estimation de pages
-  Fonction de chunking pour textes longs

4. utils/quiz_manager.py - Gestion de Quiz

Rôle: Évaluation et analyse des quiz complétés

Fonction d'évaluation:

```
def evaluate_quiz(self, quiz: Dict, user_answers: Dict) -> Dict:
    """
    Évalue un quiz complété et retourne les résultats détaillés

    Calcule:
    - Score sur 20
    - Pourcentage de réussite
    - Répartition par compétence
    - Points faibles identifiés
    - Application du barème choisi

    Returns:
    {
        "quiz_id": str,
        "score": float,           # Sur 20
        "percentage": float,      # Taux de réussite
        "earned_points": float,
        "total_points": int,
        "correct_answers": int,
        "total_questions": int,
        "competence_breakdown": Dict[str, float], # % par compétence
        "weak_areas": List[str]   # Top 5 questions ratées
    }
    """
```

Barèmes supportés:

1. **Binaire:** Tout ou rien

```

if correct:
    points = question["points"]
else:
    points = 0

```

2. Points négatifs: Pénalités pour mauvaises réponses

```

if not correct:
    points = -question["points"] * 0.25 # -25%

```

3. Partiel: Points partiels pour questions ouvertes

```

if partially_correct:
    points = question["points"] * 0.5 # 50%

```

Évaluation de questions ouvertes:

```

def _evaluate_open_question(self, user_answer: str, correct_answer: str) -> bool:
    """
    Évaluation simple par correspondance de mots-clés

    Algorithme:
    1. Extraction des mots-clés (≥4 caractères)
    2. Normalisation (lowercase, trim)
    3. Calcul du taux de correspondance
    4. Seuil: 60% des mots-clés présents = correct
    """

    keywords = set(re.findall(r'\b\w{4,}\b', correct_lower))
    user_words = set(re.findall(r'\b\w{4,}\b', user_lower))
    match_percentage = len(keywords & user_words) / len(keywords)
    return match_percentage >= 0.6

```

Fonctions Clés

1. Génération de Résumés Intelligents

Fonction: `AIGenerator.generate_summary()`

```

def generate_summary(
    self,
    text: str,
    discipline: str,
    niveau: str,
    page_count: int = 0
) -> Optional[Dict[str, Any]]:
    """
    Génère un résumé structuré avec analyse complète

    Paramètres:
        text: Texte du document (premiers 4000 chars)
        discipline: Matière académique
        niveau: Niveau d'études (L1-Doctorat)
        page_count: Nombre réel de pages

    Retour JSON:
        {
            "resume_general": str,
            "sections": List[Dict],
            "mindmap": {
                "concepts_principaux": List[str],
                "relations": List[Dict]
            },
            "timeline": List[Dict],
            "glossaire": List[Dict],
            "flashcards": List[Dict],
            "analyse": {
                "type_document": str,
                "niveau_difficulte": str,
                "mots_cles": List[str],
                "concepts_connexes": List[str],
                "temps_lecture_min": int,
                "statistiques": Dict
            }
        }
    """

```

Prompt IA optimisé:

- Contexte: Expert en {discipline} pour niveau {niveau}
- Structure: JSON strict avec toutes les sections

- Analyse: Type de document, difficulté, mots-clés
- Outils pédagogiques: Mind map, timeline, glossaire, flashcards

Gestion d'erreurs:

```
try:
    # Extraction du JSON depuis la réponse
    json_start = response_text.find("{")
    json_end = response_text.rfind("}") + 1
    json_str = response_text[json_start:json_end]
    return json.loads(json_str)
except json.JSONDecodeError:
    # Fallback vers résumé par défaut
    return self._create_default_summary()
```

2. Génération de Quiz

Fonction: AIGenerator.generate_quiz()

```

def generate_quiz(
    self,
    text: str,
    quiz_type: str,
    difficulty: str,
    num_questions: int,
    discipline: str
) -> List[Dict[str, Any]]:
    """
    Génère un quiz personnalisé basé sur le document

    Paramètres:
        text: Contenu du cours (premiers 4000 chars)
        quiz_type: Type (QCM, questions de cours, etc.)
        difficulty: Niveau (facile, moyen, difficile)
        num_questions: Nombre de questions (5-50)
        discipline: Matière académique

    Retour:
        [
            {
                "id": int,
                "type": "qcm",
                "question": str,
                "options": List[str],          # 4 options
                "correct_answer": str,
                "explication": str,
                "points": int,
                "competence": str              # Compréhension, Application, etc.
            }
        ]
    """

```

Prompt IA spécialisé:

```
prompt = f"""En tant que professeur de {discipline}, crée un quiz de type  
"{quiz_type}" avec {num_questions} questions de difficulté "{difficulty}"/>.
```

Le quiz doit inclure :

- Des questions pertinentes et académiques
- Des choix de réponse plausibles pour les QCM
- Des explications détaillées pour chaque réponse

Format JSON strict requis avec:

- id, type, question, options, correct_answer, explication, points, competence

Types de compétences : Compréhension, Application, Analyse, Mémorisation
"""

Mélange intelligent:

```
# IMPORTANT: Sauvegarder la réponse correcte AVANT le mélange  
if shuffle_options:  
    for q in questions:  
        if q["type"] == "qcm":  
            correct_answer_text = q["correct_answer"] # Texte, pas index!  
            random.shuffle(q["options"])  
            q["correct_answer"] = correct_answer_text # Reste identique
```

3. Génération de Recommandations

Fonction: AIGenerator.generate_recommendations()


```

def generate_recommendations(
    self,
    quiz_results: List[Dict[str, Any]],
    weak_areas: List[str]
) -> Optional[Dict[str, Any]]:
    """
    Génère des recommandations personnalisées basées sur les résultats

    Paramètres:
        quiz_results: Historique de tous les quiz complétés
        weak_areas: Liste des compétences/questions faibles

    Analyse effectuée:
        - Nombre total de quiz
        - Score moyen global
        - Points faibles récurrents
        - Tendances de progression

    Retour:
        {
            "points_a_revoir": [
                {"chapitre": str, "raison": str}
            ],
            "exercices_recommandes": List[str],
            "ressources": [
                {"type": str, "titre": str, "description": str}
            ],
            "strategies": List[str],
            "planning": {
                "semaine_1": List[str],
                "semaine_2": List[str]
            }
        }
    """

```

Sécurité - 5 niveaux d'erreur:

```
try:
    recommendations = ai_gen.generate_recommendations(results, weak_areas)

    # Niveau 1: Vérifier None
    if recommendations is None:
        recommendations = default_recommendations

    # Niveau 2: Fallback dans la fonction IA
    except json.JSONDecodeError:
        return self._create_default_recommendations()

    # Niveau 3: Fallback dans la page
    except Exception:
        st.session_state.recommendations = default_recommendations

    # Niveau 4: Toujours retourner quelque chose
    return recommendations or default_recommendations
```

4. Évaluation de Quiz

Fonction: QuizManager.evaluate_quiz()

```
def evaluate_quiz(self, quiz: Dict, user_answers: Dict) -> Dict:
    """
    Évalue un quiz avec calculs complexes

    Processus:
    1. Parcourir toutes les questions
    2. Comparer réponse utilisateur vs correcte
    3. Appliquer le barème choisi
    4. Agréger par compétence
    5. Identifier points faibles
    6. Calculer score final sur 20

    Normalisation des réponses:
    - Strip whitespace
    - Comparaison exacte pour QCM
    - Algorithme de mots-clés pour questions ouvertes

    Retour détaillé avec:
    - Score global
    - Pourcentage
    - Breakdown par compétence
    - Top 5 questions ratées
    """
```

Algorithme de scoring:

```
for idx, question in enumerate(questions):
    user_answer = user_answers.get(idx)

    if question["type"] == "qcm":
        # Comparaison exacte (normalisée)
        if user_answer.strip() == question["correct_answer"].strip():
            points = question["points"]
        else:
            # Application du barème
            if bareme == "Points négatifs":
                points = -question["points"] * 0.25

    # Agrégation par compétence
    competence = question["competence"]
    competence_breakdown[competence]["earned"] += points
    competence_breakdown[competence]["total"] += question["points"]

# Score final
score = (earned_points / total_points) * 20
```



Configuration et Déploiement

Installation

```
# 1. Clone le repository
git clone <repo-url>
cd python-INDIA

# 2. Créer environnement virtuel
python -m venv venv
venv\Scripts\activate # Windows
source venv/bin/activate # Linux/Mac

# 3. Installer dépendances
pip install -r requirements.txt

# 4. Configurer .env
cp .env.example .env
# Éditer .env avec vos API keys
```

Configuration .env

```
# Choix du provider IA
AI_PROVIDER=groq # Options: groq, deepseek, xai, openai, together

# API Keys (configurer selon le provider choisi)
GROQ_API_KEY=gsk_your_groq_key_here
DEEPSEEK_API_KEY=sk-your_deepseek_key
XAI_API_KEY=xai-your_xai_key
OPENAI_API_KEY=sk-your_openai_key
TOGETHER_API_KEY=your_together_key

# Configuration app
APP_ENV=development
DEBUG=True
```

Lancement

```
# Lancer l'application
streamlit run app.py

# L'app sera disponible sur http://localhost:8501
```

Déploiement Production

Streamlit Cloud:

```
# 1. Push vers GitHub
git push origin main

# 2. Sur Streamlit Cloud:
#   - Connecter le repo GitHub
#   - Ajouter les secrets (API keys) dans Settings
#   - Déployer
```

Docker (optionnel):

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
EXPOSE 8501
CMD ["streamlit", "run", "app.py"]
```



Guide d'Utilisation

Pour Étudiants

1. **Sélectionner profil "Étudiant"**
2. **Choisir discipline et niveau**
3. **Upload documents de cours (PDF/DOCX)**
4. **Analyser** → Obtenir résumés, mind maps, flashcards

5. **Générer quiz** personnalisés
6. **Passer le quiz** avec timer
7. **Consulter résultats** et recommandations
8. **Suivre progression** dans le temps

Pour Enseignants

1. **Sélectionner profil "Enseignant"**
2. **Upload supports de cours**
3. **Générer quiz** pour évaluations
4. **Configurer barèmes** et difficultés
5. **Exporter quiz** (fonctionnalité future)





Pour Chercheurs

1. **Sélectionner profil "Chercheur"**
2. **Upload articles scientifiques**
3. **Obtenir résumés structurés**
4. **Générer quiz** pour révision
5. **Analyser concepts clés**

Providers IA Recommandés

Groq (Recommandé - GRATUIT)

Avantages:

-  **100% GRATUIT** (14,400 requêtes/jour)
-  **Très rapide** (3-5 secondes par quiz)
-  Modèle: deepseek-r1-distill-llama-70b
-  Parfait pour étudiants

Configuration:

AI_PROVIDER=groq

GROQ_API_KEY=gsk_your_key

Obtenir clé: <https://console.groq.com>

DeepSeek (Très bon rapport qualité/prix)

Avantages:

- 💰 **Très bon marché** (\$0.14/1M tokens)
- ✅ Haute qualité (compétitif avec GPT-4)
- ✅ Excellent pour français académique

Configuration:

```
AI_PROVIDER=deepseek
```

```
DEEPSEEK_API_KEY=sk-your_key
```

Obtenir clé: <https://platform.deepseek.com>

xAI (Grok)

Avantages:

- ✅ Modèle Grok de X/Twitter
- ✅ Rapide et performant

Configuration:

```
AI_PROVIDER=xai
```

```
XAI_API_KEY=xai-your_key
```

Obtenir clé: <https://console.x.ai>



Statistiques du Projet

- **Lignes de code:** ~3,000+ lignes
- **Fichiers Python:** 10 fichiers
- **Pages Streamlit:** 4 pages multi-page
- **Fonctions IA:** 3 fonctions principales
- **Providers supportés:** 6 fournisseurs IA

- **Formats documents:** PDF, DOCX
- **Types de quiz:** 6 types différents
- **Niveaux:** 6 niveaux universitaires
- **Disciplines:** 10 disciplines



Sécurité et Confidentialité

- ☒ API keys stockées dans `.env` (non versionnées)
- ☒ Données utilisateur en mémoire (session Streamlit)
- ☒ Pas de stockage permanent des documents
- ☒ Compatible RGPD (données locales)
- ☒ Option Ollama pour 100% local/privé



Fonctionnalités Futures

- ☐ Stockage persistant (SQLite/PostgreSQL)
- ☐ Export PDF des résultats
- ☐ Collaboration multi-utilisateurs
- ☐ Gamification (badges, niveaux)
- ☐ Support audio/vidéo
- ☐ Intégration LMS (Moodle, Canvas)
- ☐ Application mobile
- ☐ Mode hors-ligne complet



Contribution

Ce projet est ouvert aux contributions:

1. Fork le repository
2. Créer une branche feature
3. Commit les changements
4. Push vers la branche

5. Ouvrir une Pull Request



Licence

MIT License - Libre d'utilisation pour projets personnels et commerciaux



Support

Pour questions ou bugs:

- **Issues GitHub:** [Créer une issue](#)
- **Email:** contact@quizai.fr
- **Documentation:** Ce fichier

Développé avec ❤️ pour la communauté éducative française

Dernière mise à jour: Octobre 2025