

CAHIER DES CHARGES

Kira Search Service - Microservice de Recherche Intelligente

DÉPANNAGE ET PROBLÈMES COURANTS

Problème 1 : "Failed to load embedder"

Message d'erreur :

```
ERROR: Failed to load embedder: [Errno 2] No such file or directory
```

Cause : Le modèle d'embedding n'est pas trouvé

Solutions :

1. Vérifier que le dossier existe : `ls ./models/embed/`
 2. Vérifier le chemin dans `.env` : `EMBED_MODEL_PATH=./models/embed`
 3. Re-télécharger le modèle avec le script `download_models.py`
 4. Vérifier les permissions : `chmod -R 755 ./models/`
-

Problème 2 : "MongoDB connection timeout"

Message d'erreur :

```
ERROR: MongoDB connection timeout: No servers found
```

Cause : MongoDB n'est pas accessible

Solutions :

1. Vérifier que MongoDB est démarré : `sudo systemctl status mongod`
 2. Tester la connexion : `mongosh "mongodb://localhost:27017"`
 3. Vérifier l'URI dans `.env`
 4. Vérifier le pare-feu : `sudo ufw status`
-

Problème 3 : "Memory allocation error"

Message d'erreur :

ERROR: Unable to allocate memory for array

Cause : RAM insuffisante

Solutions :

1. Fermer les applications gourmandes en mémoire
 2. Réduire `EMBEDDING_BATCH_SIZE` dans `.env` (ex: 16 au lieu de 32)
 3. Désactiver le reranking temporairement : `RERANKER_PATH=`
 4. Ajouter de la RAM au serveur
-

Problème 4 : Recherche lente (> 1 seconde)

Causes possibles :

Cause	Solution
Index trop gros	Nettoyer les documents inactifs
<code>RETRIEVE_K</code> trop élevé	Réduire à 100-150
Reranking activé	Désactiver pour tests simples
CPU surchargé	Vérifier d'autres processus

Problème 5 : Résultats non pertinents

Vérifications :

1. L'index est-il à jour ? → Appeler `/admin/rebuild`
 2. Le reranking est-il activé ? → `enable_rerank: true`
 3. Les filtres sont-ils trop restrictifs ? → Tester sans filtres
 4. La requête est-elle claire ? → Reformuler
-



MONITORING ET STATISTIQUES

Métriques importantes à surveiller

Métrique	Valeur normale	Alerte si
Temps de réponse	50-200 ms	> 500 ms
Taux d'erreur	< 1%	> 5%
Mémoire utilisée	50-70%	> 85%

Métrique	Valeur normale	Alerte si
Requêtes/seconde	10-100	> 200

Commandes de monitoring

```

bash

# Voir les logs en temps réel
tail -f ./logs/kira_search.log

# Vérifier l'utilisation mémoire
ps aux | grep python

# Voir les statistiques du service
curl http://localhost:8000/stats

# Tester la santé
watch -n 5 "curl -s http://localhost:8000/health"

```

Logs importants

Log normal (tout va bien) :

```

INFO - Search request: q="smartphone" took 45ms, 10 results
INFO - Index has 50000 documents, 48500 active

```

Log d'erreur (problème) :

```

ERROR - Embedding failed: Out of memory
ERROR - MongoDB connection lost, retrying...
WARNING - Search took 1245ms, threshold exceeded

```

SÉCURITÉ ET BONNES PRATIQUES

Recommandations de sécurité

En développement

```

env

SERVICE_HOST=127.0.0.1 # Local seulement
LOG_LEVEL=DEBUG      # Logs détaillés
MONGODB_URI=mongodb://localhost:27017 # Sans mot de passe

```

En production

```
env  
  
SERVICE_HOST=127.0.0.1 # Derrière proxy  
LOG_LEVEL=WARNING      # Moins de logs  
MONGODB_URI=mongodb://user:strongpass@host/db?ssl=true
```

Protection contre les abus

Le service intègre plusieurs protections :

Protection	Limite	Raison
Longueur requête	1000 caractères	Éviter surcharge CPU
Résultats max	100 par requête	Limiter bande passante
Candidats max	50 000	Protéger la mémoire

Proxy inverse recommandé (Nginx)

nginx

```
# Configuration Nginx  
server {  
    listen 80;  
    server_name api-search.monsite.com;  
  
    # Limite de taux  
    limit_req_zone $binary_remote_addr zone=search:10m rate=10r/s;  
    limit_req zone=search burst=20 nodelay;  
  
    location / {  
        proxy_pass http://127.0.0.1:8000;  
        proxy_set_header Host $host;  
        proxy_set_header X-Real-IP $remote_addr;  
  
        # Timeout  
        proxy_connect_timeout 5s;  
        proxy_send_timeout 10s;  
        proxy_read_timeout 30s;  
    }  
  
    # Logs  
    access_log /var/log/nginx/kira-search-access.log;  
    error_log /var/log/nginx/kira-search-error.log;  
}
```

DÉPLOIEMENT DOCKER

Dockerfile

```
dockerfile
```

```
FROM python:3.9-slim
```

```
# Variables d'environnement
```

```
ENV PYTHONUNBUFFERED=1 \
    PYTHONDONTWRITEBYTECODE=1
```

```
# Répertoire de travail
```

```
WORKDIR /app
```

```
# Installation des dépendances système
```

```
RUN apt-get update && apt-get install -y \
    curl \
    && rm -rf /var/lib/apt/lists/*
```

```
# Copie des dépendances
```

```
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
```

```
# Copie du code
```

```
COPY ..
```

```
# Création des dossiers
```

```
RUN mkdir -p logs indexes
```

```
# Exposition du port
```

```
EXPOSE 8000
```

```
# Santé check
```

```
HEALTHCHECK --interval=30s --timeout=10s --start-period=40s --retries=3 \
    CMD curl -f http://localhost:8000/health || exit 1
```

```
# Démarrage
```

```
CMD ["python", "run.py"]
```

docker-compose.yml

```
yaml
```

```
version: '3.8'

services:
  kira-search:
    build: .
    container_name: kira-search-service
    ports:
      - "8000:8000"
    volumes:
      - ./models:/app/models
      - ./indexes:/app/indexes
      - ./logs:/app/logs
    environment:
      - MONGODB_URI=mongodb://mongo:27017
      - MONGO_DB=kira
      - MONGO_COLL=posts
    depends_on:
      - mongo
    restart: unless-stopped

  mongo:
    image: mongo:5.0
    container_name: kira-mongodb
    ports:
      - "27017:27017"
    volumes:
      - mongo-data:/data/db
    restart: unless-stopped

volumes:
  mongo-data:
```

Commandes Docker

```
bash
```

```
# Construire l'image
docker-compose build

# Démarrer les services
docker-compose up -d

# Voir les logs
docker-compose logs -f kira-search

# Arrêter les services
docker-compose down

# Reconstruire l'index
docker-compose exec kira-search curl -X POST http://localhost:8000/admin/rebuild
```

PERFORMANCES ET OPTIMISATION

Benchmarks par volume

10 000 annonces

- **Mémoire** : 2 GB
- **Temps de recherche** : 20-40 ms
- **Construction index** : 3-5 minutes
- **Requêtes/sec** : 100-150

50 000 annonces

- **Mémoire** : 4 GB
- **Temps de recherche** : 40-80 ms
- **Construction index** : 15-25 minutes
- **Requêtes/sec** : 50-80

100 000 annonces

- **Mémoire** : 8 GB
- **Temps de recherche** : 80-150 ms
- **Construction index** : 30-60 minutes
- **Requêtes/sec** : 30-50

Conseils d'optimisation

Pour améliorer la vitesse

1. Utiliser un SSD rapide
2. Augmenter la RAM
3. Réduire `RETRIEVE_K` à 100
4. Désactiver le reranking pour requêtes simples

Pour économiser la RAM

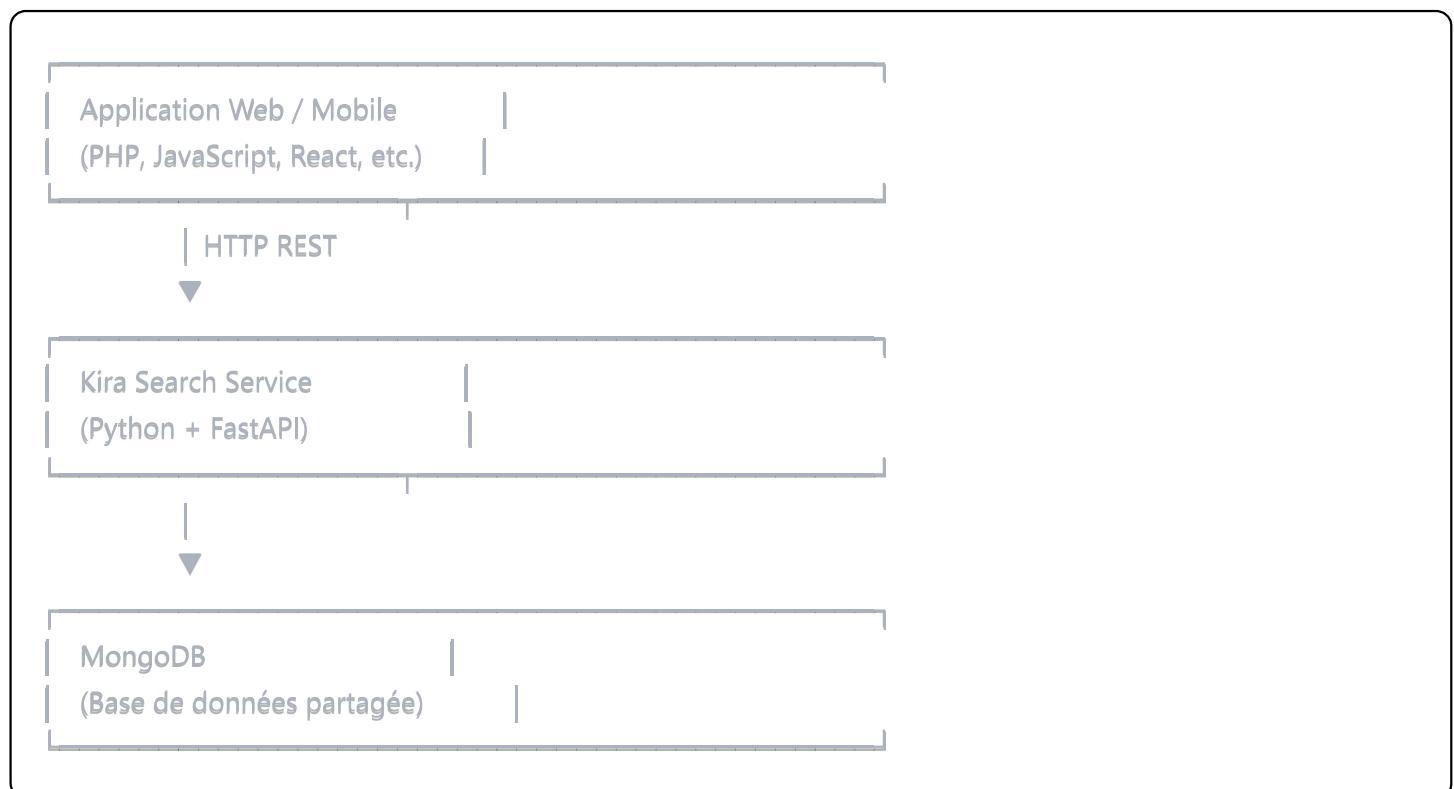
1. Réduire `EMBEDDING_BATCH_SIZE`
2. Nettoyer régulièrement les documents inactifs
3. Utiliser `memory mapping` pour l'index

Pour améliorer la pertinence

1. Activer systématiquement le reranking
2. Augmenter `RETRIEVE_K` à 300-500
3. Affiner les filtres métier

INTÉGRATION DANS VOTRE APPLICATION

Architecture recommandée



Scénarios d'utilisation

Scénario 1 : Recherche simple

```
javascript

// Frontend JavaScript
async function search(query) {
  const response = await fetch('http://api.monsite.com/search', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({
      q: query,
      topk: 10
    })
  });

  const results = await response.json();
  displayResults(results.hits);
}
```

Scénario 2 : Recherche avec filtres

```
php

// Backend PHP
function searchWithFilters($query, $city, $maxPrice) {
  $data = [
    'q' => $query,
    'city' => $city,
    'max_price' => $maxPrice,
    'topk' => 20
  ];

  $ch = curl_init('http://localhost:8000/search');
  curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($data));
  curl_setopt($ch, CURLOPT_HTTPHEADER, ['Content-Type: application/json']);
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

  $response = curl_exec($ch);
  curl_close($ch);

  return json_decode($response, true);
}
```

Scénario 3 : Pagination

```
python
```

```

# Backend Python

def search_with_pagination(query, page=1, per_page=10):
    offset = (page - 1) * per_page

    response = requests.post('http://localhost:8000/search', json={
        'q': query,
        'topk': per_page,
        'offset': offset
    })

    data = response.json()
    return {
        'results': data['hits'],
        'has_more': data['has_more'],
        'total': data['total_indexed']
    }

```

RESSOURCES ET DOCUMENTATION

Documentation officielle

Ressource	Lien
Documentation API	http://localhost:8000/docs
Schéma OpenAPI	http://localhost:8000/openapi.json
Code source	Repository Git du projet

Bibliothèques utilisées

Bibliothèque	Documentation
FastAPI	https://fastapi.tiangolo.com
SentenceTransformers	https://www.sbert.net
PyMongo	https://pymongo.readthedocs.io
NumPy	https://numpy.org/doc

Modèles IA

Modèle	Source
BGE-M3	https://huggingface.co/BAAI/bge-m3
MS-MARCO	https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-6-v2

GLOSSAIRE

Terme	Définition
Embedding	Représentation numérique (vecteur) d'un texte permettant les calculs de similarité
Reranking	Réordonnancement des résultats pour améliorer la pertinence
Similarité cosinus	Mesure mathématique de proximité entre deux vecteurs
Index vectoriel	Base de données optimisée pour la recherche de vecteurs similaires
Change Streams	Mécanisme MongoDB de notification en temps réel des modifications
Microservice	Service autonome communiquant via API REST
Endpoint	Point d'accès d'une API (URL + méthode HTTP)
Batch	Groupe de données traitées ensemble pour optimiser les performances
Token	Unité de texte (mot ou partie de mot) utilisée par les modèles IA

CHECKLIST DE DÉPLOIEMENT

Avant le déploiement

- Modèles d'IA téléchargés et placés dans `./models/`
- Fichier `.env` configuré avec les bons paramètres
- MongoDB accessible et contenant des données
- Dépendances Python installées (`pip install -r requirements.txt`)
- Dossiers créés (`logs/`, `indexes/`)
- Test local réussi (`python run.py` + `curl /health`)

Pendant le déploiement

- Index construit avec `/admin/rebuild`
- Test de recherche réussi
- Vérification des logs (pas d'erreurs)
- Test de charge basique (plusieurs requêtes simultanées)
- Monitoring configuré

Après le déploiement

- Documentation partagée avec l'équipe
- Alertes configurées (temps réponse, erreurs, mémoire)
- Sauvegardes automatiques de l'index
- Plan de mise à jour défini

SUPPORT ET MAINTENANCE

En cas de problème

1. Consulter les logs : `./logs/kira_search.log`
2. Vérifier la santé : `curl http://localhost:8000/health`
3. Consulter les stats : `curl http://localhost:8000/stats`
4. Consulter cette documentation
5. Contacter l'équipe de développement

Maintenance régulière

Tâche	Fréquence	Commande
Vérifier les logs	Quotidien	<code>tail -f logs/kira_search.log</code>
Nettoyer les logs	Hebdomadaire	<code>> logs/kira_search.log</code>
Reconstruire l'index	Mensuel	<code>POST /admin/rebuild</code>
Mettre à jour les dépendances	Trimestriel	<code>pip install -U -r requirements.txt</code>
Sauvegarder l'index	Hebdomadaire	<code>cp -r indexes/ backup/</code>

ÉVOLUTIONS FUTURES

Améliorations prévues

Court terme (1-3 mois) :

- Cache Redis pour requêtes fréquentes
- Métriques Prometheus pour monitoring avancé
- Support de plusieurs langues simultanées

Moyen terme (3-6 mois) :

- Index distribué pour très gros volumes (millions)
- Recherche par image
- Suggestions automatiques de requêtes

Long terme (6-12 mois) :

- Support GPU pour performances extrêmes
- Apprentissage continu sur les recherches utilisateur
- Multi-tenancy (plusieurs clients sur une instance)

CONCLUSION

Le Kira Search Service est une solution complète, performante et prête pour la production. Sa conception modulaire en microservice facilite son intégration dans n'importe quelle architecture existante.

Points forts du service

- Intelligence artificielle** : Compréhension du langage naturel
- Performance** : Réponses en moins de 200ms
- Scalabilité** : Supporte jusqu'à 100k+ annonces
- Fiabilité** : Gestion d'erreurs robuste
- Maintenabilité** : Code bien structuré et documenté
- Flexibilité** : Configuration via variables d'environnement

Prêt à l'emploi

Le service peut être déployé immédiatement avec :

- Documentation complète
 - Configuration simple
 - Endpoints testés et documentés
 - Monitoring intégré
 - Support de production
-

Document préparé par l'équipe Kira

Version 0.2.0 - Janvier 2025

Contact : contact@kira-search.com  INFORMATIONS GÉNÉRALES

Projet : Kira Search Service

Version : 0.2.0

Date : Janvier 2025

Type : Microservice de recherche sémantique

PRÉSENTATION DU PROJET

Qu'est-ce que le Kira Search Service ?

Le Kira Search Service est un **moteur de recherche intelligent** conçu pour une plateforme de petites annonces. Contrairement à une recherche classique qui cherche des mots exacts, ce service **comprend le sens** de ce que l'utilisateur cherche.

Problème résolu

Situation actuelle :

- Un utilisateur cherche "téléphone pas cher"
- La recherche classique ne trouve QUE les annonces contenant exactement ces mots
- Les annonces avec "smartphone abordable" ou "mobile économique" sont ignorées
- L'utilisateur manque des résultats pertinents

Avec Kira Search Service :

- L'utilisateur cherche "téléphone pas cher"
- Le service COMPREND qu'il cherche aussi : smartphone, mobile, bon marché, occasion, petit prix
- Tous les résultats pertinents sont trouvés
- L'utilisateur a de meilleurs résultats

Pourquoi ce microservice ?

Objectif principal : Améliorer drastiquement la qualité de la recherche sur la plateforme

Avantages concrets :

- Pour l'utilisateur : Trouve ce qu'il cherche plus rapidement
- Pour la plateforme : Plus de transactions réussies
- Pour le business : Meilleure satisfaction client

Architecture microservice :

- Service indépendant qui communique uniquement via API REST
- Peut être déployé séparément de l'application principale
- Facile à maintenir et à mettre à jour
- Peut être réutilisé pour d'autres projets

🧠 TECHNOLOGIES ET MODÈLES IA

Pourquoi l'Intelligence Artificielle ?

L'IA permet de comprendre le langage naturel comme un humain le ferait. Au lieu de chercher des mots identiques, elle comprend les concepts et les intentions.

1 MODÈLE D'EMBEDDING (BGE-M3)

Qu'est-ce qu'un embedding ?

Un embedding transforme du **texte en nombres** (vecteurs) que l'ordinateur peut comparer mathématiquement.

Exemple simplifié :

```
"téléphone" → [0.8, 0.2, 0.9, ...]  
"smartphone" → [0.7, 0.3, 0.8, ...]  
"voiture" → [0.1, 0.9, 0.2, ...]
```

Les vecteurs de "téléphone" et "smartphone" sont **proches** (concepts similaires).

Le vecteur de "voiture" est **éloigné** (concept différent).

Modèle choisi : BGE-M3

Nom complet : Beijing Academy of Artificial Intelligence - Multilingual Model 3

Pourquoi ce modèle ?

Critère	Détail
Multilingue	Supporte français, anglais, arabe et 100+ langues
Optimisé recherche	Spécialement entraîné pour trouver des documents
Performance	Top 3 des meilleurs modèles multilingues
Taille raisonnable	2.5 GB (pas trop lourd)
Open source	Gratuit et accessible

Caractéristiques techniques :

- **Dimension des vecteurs** : 1024 (nombre de "dimensions" pour représenter le sens)
- **Longueur maximale** : 8192 tokens (environ 6000 mots)
- **Vitesse** : 100-200 textes/seconde sur CPU standard

Avantages pour notre projet :

- Comprend le français tunisien et les expressions locales
- Gère les fautes d'orthographe courantes
- Trouve des synonymes et concepts similaires automatiquement

2 MODÈLE DE RERANKING (MS-MARCO)

Qu'est-ce que le reranking ?

Le reranking est une **deuxième étape** qui affine l'ordre des résultats en comparant directement la requête avec chaque résultat trouvé.

Processus en 2 étapes :

Étape 1 (Embedding) : Trouve 200 candidats potentiels (rapide mais approximatif)

Étape 2 (Reranking) : Classe ces 200 candidats avec précision (plus lent mais précis)

Modèle choisi : MS-MARCO MiniLM-L-6-v2

Nom complet : Microsoft Machine Reading Comprehension - Mini Language Model

Pourquoi ce modèle ?

Critère	Détail
Spécialisé	Conçu spécifiquement pour le reranking
Rapide	Version "Mini" optimisée pour la vitesse
Précis	Améliore de 15-30% la pertinence des résultats
Léger	Seulement 90 MB
Prouvé	Utilisé par de grandes entreprises

Avantages pour notre projet :

- Comprend les nuances et le contexte
- Corrige les erreurs du premier tri
- Améliore significativement la satisfaction utilisateur

3 BASE DE DONNÉES : MONGODB

Pourquoi MongoDB ?

Avantage	Explication
Flexibilité	Structure des annonces peut varier (certaines avec photos, d'autres sans)
Change Streams	Déetecte automatiquement les modifications de données
Performance	Requêtes rapides même avec millions d'annonces
Scalabilité	Facile d'ajouter plus de serveurs si besoin
Filtrage	Filtres complexes (ville + catégorie + prix) très rapides

Structure d'une annonce dans MongoDB

javascript

```
{  
  "_id": "507f1f77bcf86cd799439011",      // Identifiant unique  
  "title": "iPhone 13 Pro 128GB",          // Titre de l'annonce  
  "description": "Excellent état, avec boîte", // Description complète  
  "price": 650.0,                          // Prix en dinars  
  "city": "Tunis",                         // Ville  
  "category": "Téléphones",                // Catégorie  
  "url": "https://site.com/annonce/123",    // Lien vers l'annonce  
  "is_active": true                      // Annonce active ou non  
}
```

4 API : FASTAPI + UVICORN

Pourquoi FastAPI ?

FastAPI est un framework Python moderne pour créer des APIs REST.

Avantage	Bénéfice
Documentation auto	Interface web automatique pour tester l'API
Validation auto	Vérifie que les données reçues sont correctes
Performance	Aussi rapide que Node.js ou Go
Type hints	Code plus clair et moins d'erreurs
Async	Gère plusieurs requêtes simultanément

Uvicorn est le serveur qui fait tourner FastAPI (équivalent de Apache/Nginx pour Python).

STRUCTURE DU PROJET

Vue d'ensemble de l'arborescence

```
kira-search-service/  
|  
|   app/           # Code principal du service  
|   |   __init__.py # Initialisation du package Python  
|   |   main.py     # Point d'entrée de l'API  
|   |   config.py   # Configuration et variables  
|   |   embedding.py # Gestion des embeddings  
|   |   reranker.py # Gestion du reranking
```

```

|   |   └── indexer.py      # Gestion de l'index vectoriel
|   |   └── mongo.py       # Connexion MongoDB
|   └── utils.py          # Fonctions utilitaires

|   └── models/           # Modèles d'intelligence artificielle
|       └── embed/         # Modèle d'embedding (2.5 GB)
|           └── reranker/  # Modèle de reranking (90 MB)

|   └── indexes/          # Index vectoriel persisté
|       └── vectors.npy    # Vecteurs de tous les documents
|       └── ids.json        # Correspondance ID ↔ position
|       └── deleted.npy     # Masque des documents supprimés

|   └── logs/              # Fichiers de logs
|       └── kira_search.log # Journal d'activité du service

└── .env                  # Variables d'environnement
└── requirements.txt      # Dépendances Python
└── run.py                # Script de démarrage
└── README.md             # Documentation

```

Détail des fichiers principaux

app/main.py - Cœur de l'API

Rôle : Définit tous les endpoints (routes) de l'API

Contenu :

- Configuration de FastAPI
- Endpoint `/search` : recherche principale
- Endpoint `/admin/rebuild` : reconstruction de l'index
- Endpoint `/stats` : statistiques
- Endpoint `/health` : vérification du service
- Gestion des erreurs

Lignes de code : ~400 lignes

app/config.py - Configuration

Rôle : Centralise toutes les variables de configuration

Variables gérées :

python

```
MONGODB_URI      # Adresse de connexion MongoDB
EMBED_MODEL_PATH # Chemin du modèle d'embedding
RERANKER_PATH    # Chemin du modèle de reranking
INDEX_DIR        # Dossier de l'index
RETRIEVE_K       # Nombre de candidats à récupérer
FINAL_TOPK       # Nombre de résultats finaux
```

Lignes de code : ~40 lignes

app/embedding.py - Service d'embedding

Rôle : Transforme du texte en vecteurs

Fonctions principales :

- `encode_texts()` : Convertit plusieurs textes en vecteurs (batch)
- `encode_one()` : Convertit un seul texte en vecteur

Processus :

```
Texte → Nettoyage → Tokenisation → Modèle IA → Vecteur normalisé
```

Lignes de code : ~80 lignes

app/reranker.py - Service de reranking

Rôle : Affine l'ordre des résultats

Fonctions principales :

- `rerank()` : Compare la requête avec chaque résultat et donne un score précis
- `is_available()` : Vérifie si le modèle est chargé

Processus :

```
[Requête + Résultat 1] → Modèle → Score 0.89
[Requête + Résultat 2] → Modèle → Score 0.75
[Requête + Résultat 3] → Modèle → Score 0.92
→ Tri par score décroissant
```

Lignes de code : ~120 lignes

app/indexer.py - Gestion de l'index

Rôle : Stocke et recherche dans les vecteurs

Fonctions principales :

- `add_many()` : Ajoute plusieurs documents à l'index
- `upsert_one()` : Ajoute ou met à jour un document
- `delete_one()` : Marque un document comme supprimé (suppression logique)
- `search_rows()` : Recherche les documents les plus similaires
- `save()` / `load()` : Persistance sur disque

Optimisations :

- Suppression logique (pas de réorganisation de l'index)
- Calcul vectoriel optimisé avec NumPy
- Sauvegarde incrémentale

Lignes de code : ~350 lignes

app/mongo.py - Interface MongoDB

Rôle : Communication avec la base de données

Fonctions principales :

- `get_collection()` : Établit la connexion MongoDB
- `iter_active_docs()` : Itère sur toutes les annonces actives
- `get_docs_by_ids()` : Récupère des documents par leurs IDs
- `candidate_ids_from_filters()` : Applique les filtres (ville, prix, catégorie)
- `get_collection_stats()` : Statistiques de la base

Lignes de code : ~250 lignes

app/utils.py - Fonctions utilitaires

Rôle : Fonctions réutilisables

Fonctions principales :

- `doc_to_text()` : Convertit une annonce en texte pour l'embedding

- `clean_query()` : Nettoie une requête utilisateur
- `validate_price_range()` : Valide une fourchette de prix

Exemple de conversion :

```
python

# Annonce MongoDB
{
    "title": "iPhone 13",
    "price": 650,
    "city": "Tunis"
}

# Texte pour embedding
"Titre: iPhone 13\nVille: Tunis\nprix: 650"
```

Lignes de code : ~150 lignes

.env - Variables d'environnement

Rôle : Configuration sans toucher au code

Contenu type :

```
env

# Chemins des modèles
EMBED_MODEL_PATH=./models/embed
RERANKER_PATH=./models/reranker

# MongoDB
MONGODB_URI=mongodb://localhost:27017
MONGO_DB=kira
MONGO_COLL=posts

# Service
SERVICE_HOST=0.0.0.0
SERVICE_PORT=8000

# Paramètres de recherche
RETRIEVE_K=200
FINAL_TOPK=10
```

run.py - Script de démarrage

Rôle : Lance le service

Processus :

1. Charge la configuration
2. Configure les logs
3. Démarrre le serveur Uvicorn
4. Gère l'arrêt propre

Utilisation :

```
bash
python run.py
```

Lignes de code : ~50 lignes

requirements.txt - Dépendances

Rôle : Liste toutes les bibliothèques Python nécessaires

Principales dépendances :

```
fastapi==0.116.1      # Framework API
sentence-transformers==5.1.0 # Modèles d'embedding
torch==2.8.0          # Framework deep learning
pymongo==4.14.1        # Driver MongoDB
numpy==2.3.3           # Calculs vectoriels
faiss-cpu==1.12.0       # Recherche vectorielle optimisée
```

💡 ENDPOINTS DE L'API

Vue d'ensemble

Le service expose 4 endpoints principaux :

Endpoint	Méthode	Rôle
/search	POST	Recherche principale
/admin/rebuild	POST	Reconstruction de l'index
/stats	GET	Statistiques détaillées
/health	GET	Vérification de santé
◀		▶

1 ENDPOINT : /search

Méthode : POST

Rôle : Effectue une recherche sémantique dans les annonces

Paramètres de la requête

Paramètre	Type	Obligatoire	Description
q	string	✓ Oui	Texte de recherche (1-1000 caractères)
topk	integer	✗ Non	Nombre de résultats (défaut: 10, max: 100)
retrieve_k	integer	✗ Non	Nombre de candidats (défaut: 200, max: 1000)
offset	integer	✗ Non	Décalage pour pagination (défaut: 0)
city	string	✗ Non	Filtrer par ville
category	string	✗ Non	Filtrer par catégorie
min_price	float	✗ Non	Prix minimum
max_price	float	✗ Non	Prix maximum
is_active	boolean	✗ Non	Annonces actives uniquement (défaut: true)
enable_rerank	boolean	✗ Non	Activer le reranking (défaut: true)
◀		▶	

Exemple de requête

bash

```
curl -X POST http://localhost:8000/search \
-H "Content-Type: application/json" \
-d '{
  "q": "smartphone pas cher",
  "topk": 10,
  "city": "Tunis",
  "max_price": 500,
  "enable_rerank": true
}'
```

Réponse

json

```
{  
  "total_indexed": 50000,  
  "took_ms": 87,  
  "hits": [  
    {  
      "id": "507f1f77bcf86cd799439011",  
      "score": 0.89,  
      "title": "iPhone 13 Pro 128GB Excellent état",  
      "price": 450.0,  
      "city": "Tunis",  
      "category": "Téléphones",  
      "url": "https://monsite.com/annonce/12345"  
    },  
    {  
      "id": "507f1f77bcf86cd799439012",  
      "score": 0.85,  
      "title": "Samsung Galaxy S21 comme neuf",  
      "price": 380.0,  
      "city": "Tunis",  
      "category": "Téléphones",  
      "url": "https://monsite.com/annonce/67890"  
    }  
  ],  
  "reranked": true,  
  "offset": 0,  
  "has_more": false  
}
```

Explication de la réponse

Champ	Description
total_indexed	Nombre total de documents dans l'index
took_ms	Temps de traitement en millisecondes
hits	Liste des résultats trouvés
score	Score de pertinence (0.0 à 1.0)
reranked	Si le reranking a été appliqué
has_more	S'il reste des résultats (pour pagination)

2 ENDPOINT : /admin/rebuild

Méthode : POST

Rôle : Reconstruit l'index depuis MongoDB

Paramètres

Paramètre	Type	Description
batch_size	integer	Taille des batches (défaut: 128, max: 1000)

Quand l'utiliser ?

- Premier démarrage du service
- Après importation massive de données
- Si l'index semble corrompu
- Pour forcer une resynchronisation complète

Exemple de requête

```
bash
```

```
curl -X POST http://localhost:8000/admin/rebuild?batch_size=256
```

Réponse

json
{ "ok": true, "indexed": 50000, "processed": 50000, "took_ms": 1245000 }

Temps estimés

Volume	Temps approximatif
10 000 annonces	2-5 minutes
50 000 annonces	10-20 minutes
100 000 annonces	30-60 minutes

3 ENDPOINT : /stats

Méthode : GET

Rôle : Récupère des statistiques détaillées

Exemple de requête

```
bash
```

```
curl http://localhost:8000/stats
```

Réponse

```
json
```

```
{
  "service": {
    "embedder_ready": true,
    "reranker_available": true,
    "change_streams_enabled": false,
    "retrieve_k": 200,
    "final_topk": 10
  },
  "index": {
    "total_docs": 50000,
    "active_docs": 48500,
    "deleted_docs": 1500,
    "vector_dim": 1024,
    "memory_usage_mb": 1250.5
  },
  "mongodb": {
    "total_documents": 52000,
    "active_documents": 48500,
    "average_price": 325.50,
    "unique_cities": 45,
    "unique_categories": 28
  },
  "version": "0.2.0"
}
```

4 ENDPOINT : /health

Méthode : GET

Rôle : Vérification rapide du statut du service

Exemple de requête

```
bash
curl http://localhost:8000/health
```

Réponse (service OK)

```
json
{
  "ok": true,
  "indexed": 50000,
  "embedder_ready": true,
  "reranker_ready": true,
  "mongodb_ready": true
}
```

Utilisation

- Monitoring automatique (Kubernetes, Docker)
- Load balancers
- Scripts de surveillance
- Tests d'intégration

VARIABLES D'ENVIRONNEMENT

Configuration complète du fichier .env

Section 1 : Chemins des modèles

```
env
# Chemin vers le modèle d'embedding (BGE-M3)
EMBED_MODEL_PATH=./models/embed

# Chemin vers le modèle de reranking (MS-MARCO)
# Laisser vide pour désactiver le reranking
RERANKER_PATH=./models/reranker
```

Explication :

- Ces chemins indiquent où trouver les modèles d'IA
- Les modèles doivent être téléchargés séparément (gros fichiers)

- Si **RERANKER_PATH** est vide, le reranking sera désactivé
-

Section 2 : Index vectoriel

env

Dossier où l'index sera sauvegardé

INDEX_DIR=./indexes

Explication :

- L'index contient tous les vecteurs des annonces
 - Crée automatiquement au premier démarrage
 - Taille approximative : 50 MB par 10 000 annonces
-

Section 3 : MongoDB

env

URI de connexion à MongoDB

MONGODB_URI=mongodb://localhost:27017

Nom de la base de données

MONGO_DB=kira

Nom de la collection contenant les annonces

MONGO_COLL=posts

Activer la synchronisation automatique (Change Streams)

ENABLE_CHANGE_STREAMS=false

Explication :

Variable	Exemple	Description
MONGODB_URI	mongodb://localhost:27017	Adresse du serveur MongoDB
	mongodb://user:pass@host:27017	Avec authentification
	mongodb://host1:27017,host2:27017	Cluster MongoDB
MONGO_DB	kira	Nom de votre base de données
MONGO_COLL	posts	Collection contenant les annonces
ENABLE_CHANGE_STREAMS	true/false	Synchronisation temps réel

Change Streams :

- Si `true` : Le service détecte automatiquement les ajouts/modifications/suppressions
- Si `false` : Il faut appeler `/admin/rebuild` manuellement pour synchroniser

Section 4 : Réseau du service

```
env
```

```
# Adresse d'écoute du service
```

```
SERVICE_HOST=0.0.0.0
```

```
# Port d'écoute
```

```
SERVICE_PORT=8000
```

Explication :

Variable	Valeur	Usage
<code>SERVICE_HOST</code>	<code>0.0.0.0</code>	Écoute sur toutes les interfaces (production)
	<code>127.0.0.1</code>	Écoute locale uniquement (développement)
<code>SERVICE_PORT</code>	<code>8000</code>	Port standard (peut être changé)

Section 5 : Paramètres de recherche

```
env
```

```
# Nombre de candidats à récupérer lors de la recherche vectorielle
```

```
RETRIEVE_K=200
```

```
# Nombre de résultats finaux à retourner par défaut
```

```
FINAL_TOPK=10
```

Explication :

`RETRIEVE_K` : Nombre de documents récupérés avant le reranking

- Plus grand = meilleure qualité mais plus lent
- Recommandé : 100-500
- Notre choix : 200 (bon compromis)

`FINAL_TOPK` : Nombre de résultats retournés à l'utilisateur

- Généralement 10-20 résultats
- Peut être modifié par requête avec le paramètre `topk`

Section 6 : Performance

env

Taille des batches pour l'embedding

`EMBEDDING_BATCH_SIZE=32`

Longueur maximale d'une requête (caractères)

`MAX_QUERY_LENGTH=1000`

Nombre maximum de candidats lors du filtrage MongoDB

`MAX_CANDIDATES_FILTER=50000`

Explication :

Variable	Impact	Recommandation
<code>EMBEDDING_BATCH_SIZE</code>	Mémoire et vitesse	16-64 selon RAM
<code>MAX_QUERY_LENGTH</code>	Protection contre abus	500-2000
<code>MAX_CANDIDATES_FILTER</code>	Limite des filtres MongoDB	20000-100000

Section 7 : Logging

env

Niveau de détail des logs

`LOG_LEVEL=INFO`

Niveaux disponibles :

Niveau	Détail	Usage
<code>DEBUG</code>	Maximum	Développement, dépannage
<code>INFO</code>	Standard	Production normale
<code>WARNING</code>	Avertissements	Production optimisée
<code>ERROR</code>	Erreurs uniquement	Production minimale

Exemples de configurations

Configuration DÉVELOPPEMENT

```
env  
  
EMBED_MODEL_PATH=./models/embed  
RERANKER_PATH=./models/reranker  
INDEX_DIR=./indexes  
  
MONGODB_URI=mongodb://localhost:27017  
MONGO_DB=kira_dev  
MONGO_COLL=posts  
ENABLE_CHANGE_STREAMS=true  
  
SERVICE_HOST=127.0.0.1  
SERVICE_PORT=8000  
  
RETRIEVE_K=100  
FINAL_TOPK=5  
  
EMBEDDING_BATCH_SIZE=16  
LOG_LEVEL=DEBUG
```

Configuration PRODUCTION

```
env  
  
EMBED_MODEL_PATH=/opt/kira/models/embed  
RERANKER_PATH=/opt/kira/models/reranker  
INDEX_DIR=/var/kira/indexes  
  
MONGODB_URI=mongodb://user:pass@mongo-cluster:27017/kira?ssl=true  
MONGO_DB=kira_prod  
MONGO_COLL=posts  
ENABLE_CHANGE_STREAMS=true  
  
SERVICE_HOST=127.0.0.1  
SERVICE_PORT=8000  
  
RETRIEVE_K=200  
FINAL_TOPK=10  
  
EMBEDDING_BATCH_SIZE=64  
MAX_QUERY_LENGTH=500  
LOG_LEVEL=WARNING
```

GUIDE DE DÉMARRAGE RAPIDE

Étape 1 : Installation

```
bash

# Cloner le projet
git clone https://github.com/votre-org/kira-search-service
cd kira-search-service

# Créer l'environnement virtuel
python -m venv .venv

# Activer l'environnement
# Sur Linux/Mac :
source .venv/bin/activate
# Sur Windows :
.venv\Scripts\activate

# Installer les dépendances
pip install -r requirements.txt
```

Étape 2 : Configuration

```
bash

# Copier le fichier de configuration
cp .env.example .env

# Éditer avec vos paramètres
nano .env
```

Étape 3 : Télécharger les modèles

```
python
```

```
# Créer un script download_models.py
from sentence_transformers import SentenceTransformer, CrossEncoder

# Télécharger le modèle d'embedding
print("Téléchargement du modèle d'embedding...")
embedder = SentenceTransformer('BAAI/bge-m3')
embedder.save('./models/embed')

# Télécharger le modèle de reranking
print("Téléchargement du modèle de reranking...")
reranker = CrossEncoder('cross-encoder/ms-marco-MiniLM-L-6-v2')
reranker.save('./models/reranker')

print("Téléchargement terminé !")
```

bash

```
# Exécuter le script
python download_models.py
```

Étape 4 : Démarrer le service

bash

```
python run.py
```

Étape 5 : Vérifier le fonctionnement

bash

```
# Test de santé
curl http://localhost:8000/health

# Devrait retourner :
# {"ok": true, "indexed": 0, ...}
```

Étape 6 : Construire l'index

bash

```
# Si vous avez des données dans MongoDB
curl -X POST http://localhost:8000/admin/rebuild

# Sinon, continuez au test de recherche
```

Étape 7 : Test de recherche

```
bash
```

```
curl -X POST http://localhost:8000/search \  
-H "Content-Type: application/json" \  
-d '{"q": "test"}'
```

CAHIER DES CHARGES

Kira Search Service - Microservice de Recherche Intelligente

INFORMATIONS GÉNÉRALES

Projet : Kira Search Service

Version : 0.2.0

Date : Janvier 2025

Type : Microservice de recherche sémantique

PRÉSENTATION DU PROJET

Qu'est-ce que le Kira Search Service ?

Le Kira Search Service est un **moteur de recherche intelligent** conçu pour une plateforme de petites annonces. Contrairement à une recherche classique qui cherche des mots exacts, ce service **comprend le sens** de ce que l'utilisateur cherche.

Problème résolu

Situation actuelle :

- Un utilisateur cherche "téléphone pas cher"
- La recherche classique ne trouve QUE les annonces contenant exactement ces mots
- Les annonces avec "smartphone abordable" ou "mobile économique" sont ignorées
- L'utilisateur manque des résultats pertinents

Avec Kira Search Service :

- L'utilisateur cherche "téléphone pas cher"
- Le service COMPREND qu'il cherche aussi : smartphone, mobile, bon marché, occasion, petit prix
- Tous les résultats pertinents sont trouvés
- L'utilisateur a de meilleurs résultats

Pourquoi ce microservice ?

Objectif principal : Améliorer drastiquement la qualité de la recherche sur la plateforme

Avantages concrets :

- Pour l'utilisateur : Trouve ce qu'il cherche plus rapidement

- Pour la plateforme : Plus de transactions réussies
- Pour le business : Meilleure satisfaction client

Architecture microservice :

- Service indépendant qui communique uniquement via API REST
 - Peut être déployé séparément de l'application principale
 - Facile à maintenir et à mettre à jour
 - Peut être réutilisé pour d'autres projets
-

TECHNOLOGIES ET MODÈLES IA

Pourquoi l'Intelligence Artificielle ?

L'IA permet de **comprendre le langage naturel** comme un humain le ferait. Au lieu de chercher des mots identiques, elle comprend les concepts et les intentions.

1 MODELE D'EMBEDDING (BGE-M3)

Qu'est-ce qu'un embedding ?

Un embedding transforme du **texte en nombres** (vecteurs) que l'ordinateur peut comparer mathématiquement.

Exemple simplifié :

```
"téléphone" → [0.8, 0.2, 0.9, ...]  
"smartphone" → [0.7, 0.3, 0.8, ...]  
"voiture" → [0.1, 0.9, 0.2, ...]
```

Les vecteurs de "téléphone" et "smartphone" sont **proches** (concepts similaires).

Le vecteur de "voiture" est **éloigné** (concept différent).

Modèle choisi : BGE-M3

Nom complet : Beijing Academy of Artificial Intelligence - Multilingual Model 3

Pourquoi ce modèle ?

Critère	Détail
Multilingue	Supporte français, anglais, arabe et 100+ langues
Optimisé recherche	Spécialement entraîné pour trouver des documents
Performance	Top 3 des meilleurs modèles multilingues
Taille raisonnable	2.5 GB (pas trop lourd)
Open source	Gratuit et accessible

Caractéristiques techniques :

- **Dimension des vecteurs** : 1024 (nombre de "dimensions" pour représenter le sens)
- **Longueur maximale** : 8192 tokens (environ 6000 mots)
- **Vitesse** : 100-200 textes/seconde sur CPU standard

Avantages pour notre projet :

- Comprend le français tunisien et les expressions locales
- Gère les fautes d'orthographe courantes
- Trouve des synonymes et concepts similaires automatiquement

2 MODÈLE DE RERANKING (MS-MARCO)

Qu'est-ce que le reranking ?

Le reranking est une **deuxième étape** qui affine l'ordre des résultats en comparant directement la requête avec chaque résultat trouvé.

Processus en 2 étapes :

Étape 1 (Embedding) : Trouve 200 candidats potentiels (rapide mais approximatif)

Étape 2 (Reranking) : Classe ces 200 candidats avec précision (plus lent mais précis)

Modèle choisi : MS-MARCO MiniLM-L-6-v2

Nom complet : Microsoft Machine Reading Comprehension - Mini Language Model

Pourquoi ce modèle ?

Critère	Détail
Spécialisé	Conçu spécifiquement pour le reranking
Rapide	Version "Mini" optimisée pour la vitesse
Précis	Améliore de 15-30% la pertinence des résultats
Léger	Seulement 90 MB
Prouvé	Utilisé par de grandes entreprises

Avantages pour notre projet :

- Comprend les nuances et le contexte
- Corrige les erreurs du premier tri
- Améliore significativement la satisfaction utilisateur

3 BASE DE DONNÉES : MONGODB

Pourquoi MongoDB ?

Avantage	Explication
Flexibilité	Structure des annonces peut varier (certaines avec photos, d'autres sans)
Change Streams	Déetecte automatiquement les modifications de données
Performance	Requêtes rapides même avec millions d'annonces
Scalabilité	Facile d'ajouter plus de serveurs si besoin
Filtrage	Filtres complexes (ville + catégorie + prix) très rapides

Structure d'une annonce dans MongoDB

javascript

```
{
  "_id": "507f1f77bcf86cd799439011",      // Identifiant unique
  "title": "iPhone 13 Pro 128GB",           // Titre de l'annonce
  "description": "Excellent état, avec boîte", // Description complète
  "price": 650.0,                          // Prix en dinars
  "city": "Tunis",                         // Ville
  "category": "Téléphones",                // Catégorie
  "url": "https://site.com/annonce/123",    // Lien vers l'annonce
  "is_active": true                        // Annonce active ou non
}
```

4 API : FASTAPI + UVICORN

Pourquoi FastAPI ?

FastAPI est un framework Python moderne pour créer des APIs REST.

Avantage	Bénéfice
Documentation auto	Interface web automatique pour tester l'API
Validation auto	Vérifie que les données reçues sont correctes
Performance	Aussi rapide que Node.js ou Go
Type hints	Code plus clair et moins d'erreurs
Async	Gère plusieurs requêtes simultanément

Uvicorn est le serveur qui fait tourner FastAPI (équivalent de Apache/Nginx pour Python).

STRUCTURE DU PROJET

Vue d'ensemble de l'arborescence

kira-search-service/	
app/	# Code principal du service
_ __init__.py	# Initialisation du package Python
_ main.py	# Point d'entrée de l'API
_ config.py	# Configuration et variables
_ embedding.py	# Gestion des embeddings
_ reranker.py	# Gestion du reranking
_ indexer.py	# Gestion de l'index vectoriel
_ mongo.py	# Connexion MongoDB
_ utils.py	# Fonctions utilitaires
models/	# Modèles d'intelligence artificielle
embed/	# Modèle d'embedding (2.5 GB)
reranker/	# Modèle de reranking (90 MB)
indexes/	# Index vectoriel persisté
vectors.npy	# Vecteurs de tous les documents
ids.json	# Correspondance ID ↔ position
deleted.npy	# Masque des documents supprimés
logs/	# Fichiers de logs
kira_search.log	# Journal d'activité du service
.env	# Variables d'environnement

```
└── requirements.txt      # Dépendances Python  
└── run.py               # Script de démarrage  
└── README.md            # Documentation
```

Détail des fichiers principaux

app/main.py - Cœur de l'API

Rôle : Définit tous les endpoints (routes) de l'API

Contenu :

- Configuration de FastAPI
- Endpoint `/search` : recherche principale
- Endpoint `/admin/rebuild` : reconstruction de l'index
- Endpoint `/stats` : statistiques
- Endpoint `/health` : vérification du service
- Gestion des erreurs

Lignes de code : ~400 lignes

app/config.py - Configuration

Rôle : Centralise toutes les variables de configuration

Variables gérées :

```
python  
  
MONGODB_URI      # Adresse de connexion MongoDB  
EMBED_MODEL_PATH # Chemin du modèle d'embedding  
RERANKER_PATH    # Chemin du modèle de reranking  
INDEX_DIR        # Dossier de l'index  
RETRIEVE_K       # Nombre de candidats à récupérer  
FINAL_TOPK       # Nombre de résultats finaux
```

Lignes de code : ~40 lignes

app/embedding.py - Service d'embedding

Rôle : Transforme du texte en vecteurs

Fonctions principales :

- `encode_texts()` : Convertit plusieurs textes en vecteurs (batch)
- `encode_one()` : Convertit un seul texte en vecteur

Processus :

```
Texte → Nettoyage → Tokenisation → Modèle IA → Vecteur normalisé
```

Lignes de code : ~80 lignes

app/reranker.py - Service de reranking

Rôle : Affine l'ordre des résultats

Fonctions principales :

- `rerank()` : Compare la requête avec chaque résultat et donne un score précis
- `is_available()` : Vérifie si le modèle est chargé

Processus :

```
[Requête + Résultat 1] → Modèle → Score 0.89  
[Requête + Résultat 2] → Modèle → Score 0.75  
[Requête + Résultat 3] → Modèle → Score 0.92  
→ Tri par score décroissant
```

Lignes de code : ~120 lignes

app/indexer.py - Gestion de l'index

Rôle : Stocke et recherche dans les vecteurs

Fonctions principales :

- `add_many()` : Ajoute plusieurs documents à l'index
- `upsert_one()` : Ajoute ou met à jour un document
- `delete_one()` : Marque un document comme supprimé (suppression logique)
- `search_rows()` : Recherche les documents les plus similaires
- `save()` / `load()` : Persistance sur disque

Optimisations :

- Suppression logique (pas de réorganisation de l'index)
- Calcul vectoriel optimisé avec NumPy
- Sauvegarde incrémentale

Lignes de code : ~350 lignes

app/mongo.py - Interface MongoDB

Rôle : Communication avec la base de données

Fonctions principales :

- `get_collection()` : Établit la connexion MongoDB
- `iter_active_docs()` : Itère sur toutes les annonces actives
- `get_docs_by_ids()` : Récupère des documents par leurs IDs
- `candidate_ids_from_filters()` : Applique les filtres (ville, prix, catégorie)
- `get_collection_stats()` : Statistiques de la base

Lignes de code : ~250 lignes

app/utils.py - Fonctions utilitaires

Rôle : Fonctions réutilisables

Fonctions principales :

- `doc_to_text()` : Convertit une annonce en texte pour l'embedding
- `clean_query()` : Nettoie une requête utilisateur
- `validate_price_range()` : Valide une fourchette de prix

Exemple de conversion :

```
python
```

```

# Annonce MongoDB
{
  "title": "iPhone 13",
  "price": 650,
  "city": "Tunis"
}

# Texte pour embedding
"Titre: iPhone 13\nVille: Tunis\nprix: 650"

```

Lignes de code : ~150 lignes

.env - Variables d'environnement

Rôle : Configuration sans toucher au code

Contenu type :

```

env

# Chemins des modèles
EMBED_MODEL_PATH=./models/embed
RERANKER_PATH=./models/reranker

# MongoDB
MONGODB_URI=mongodb://localhost:27017
MONGO_DB=kira
MONGO_COLL=posts

# Service
SERVICE_HOST=0.0.0.0
SERVICE_PORT=8000

# Paramètres de recherche
RETRIEVE_K=200
FINAL_TOPK=10

```

run.py - Script de démarrage

Rôle : Lance le service

Processus :

1. Charge la configuration

2. Configure les logs
3. Démarrer le serveur Unicorn
4. Gère l'arrêt propre

Utilisation :

```
bash  
python run.py
```

Lignes de code : ~50 lignes

requirements.txt - Dépendances

Rôle : Liste toutes les bibliothèques Python nécessaires

Principales dépendances :

```
fastapi==0.116.1      # Framework API  
sentence-transformers==5.1.0 # Modèles d'embedding  
torch==2.8.0          # Framework deep learning  
pymongo==4.14.1       # Driver MongoDB  
numpy==2.3.3          # Calculs vectoriels  
faiss-cpu==1.12.0     # Recherche vectorielle optimisée
```

🔌 ENDPOINTS DE L'API

Vue d'ensemble

Le service expose 4 endpoints principaux :

Endpoint	Méthode	Rôle
/search	POST	Recherche principale
/admin/rebuild	POST	Reconstruction de l'index
/stats	GET	Statistiques détaillées
/health	GET	Vérification de santé

1 ENDPOINT : /search

Méthode : POST

Rôle : Effectue une recherche sémantique dans les annonces

Paramètres de la requête

Paramètre	Type	Obligatoire	Description
<code>q</code>	string	<input checked="" type="checkbox"/> Oui	Texte de recherche (1-1000 caractères)
<code>topk</code>	integer	<input type="checkbox"/> Non	Nombre de résultats (défaut: 10, max: 100)
<code>retrieve_k</code>	integer	<input type="checkbox"/> Non	Nombre de candidats (défaut: 200, max: 1000)
<code>offset</code>	integer	<input type="checkbox"/> Non	Décalage pour pagination (défaut: 0)
<code>city</code>	string	<input type="checkbox"/> Non	Filtrer par ville
<code>category</code>	string	<input type="checkbox"/> Non	Filtrer par catégorie
<code>min_price</code>	float	<input type="checkbox"/> Non	Prix minimum
<code>max_price</code>	float	<input type="checkbox"/> Non	Prix maximum
<code>is_active</code>	boolean	<input type="checkbox"/> Non	Annonces actives uniquement (défaut: true)
<code>enable_rerank</code>	boolean	<input type="checkbox"/> Non	Activer le reranking (défaut: true)

Exemple de requête

bash

```
curl -X POST http://localhost:8000/search \
-H "Content-Type: application/json" \
-d '{
  "q": "smartphone pas cher",
  "topk": 10,
  "city": "Tunis",
  "max_price": 500,
  "enable_rerank": true
}'
```

Réponse

json

```
{
  "total_indexed": 50000,
  "took_ms": 87,
  "hits": [
    {
      "id": "507f1f77bcf86cd799439011",
      "score": 0.89,
      "title": "iPhone 13 Pro 128GB Excellent état",
      "price": 450.0,
      "city": "Tunis",
      "category": "Téléphones",
      "url": "https://monsite.com/annonce/12345"
    },
    {
      "id": "507f1f77bcf86cd799439012",
      "score": 0.85,
      "title": "Samsung Galaxy S21 comme neuf",
      "price": 380.0,
      "city": "Tunis",
      "category": "Téléphones",
      "url": "https://monsite.com/annonce/67890"
    }
  ],
  "reranked": true,
  "offset": 0,
  "has_more": false
}
```

Explication de la réponse

Champ	Description
total_indexed	Nombre total de documents dans l'index
took_ms	Temps de traitement en millisecondes
hits	Liste des résultats trouvés
score	Score de pertinence (0.0 à 1.0)
reranked	Si le reranking a été appliqué
has_more	S'il reste des résultats (pour pagination)

2) ENDPOINT : `/admin/rebuild`

Méthode : `POST`

Rôle : Reconstruit l'index depuis MongoDB

Paramètres

Paramètre	Type	Description
batch_size	integer	Taille des batches (défaut: 128, max: 1000)

Quand l'utiliser ?

- Premier démarrage du service
- Après importation massive de données
- Si l'index semble corrompu
- Pour forcer une resynchronisation complète

Exemple de requête

bash

```
curl -X POST http://localhost:8000/admin/rebuild?batch_size=256
```

Réponse

json

```
{
  "ok": true,
  "indexed": 50000,
  "processed": 50000,
  "took_ms": 1245000
}
```

Temps estimés

Volume	Temps approximatif
10 000 annonces	2-5 minutes
50 000 annonces	10-20 minutes
100 000 annonces	30-60 minutes

3) ENDPOINT : `/stats`

Méthode : `GET`

Rôle : Récupère des statistiques détaillées

Exemple de requête

```
bash
```

```
curl http://localhost:8000/stats
```

Réponse

```
json
```

```
{
  "service": {
    "embedder_ready": true,
    "reranker_available": true,
    "change_streams_enabled": false,
    "retrieve_k": 200,
    "final_topk": 10
  },
  "index": {
    "total_docs": 50000,
    "active_docs": 48500,
    "deleted_docs": 1500,
    "vector_dim": 1024,
    "memory_usage_mb": 1250.5
  },
  "mongodb": {
    "total_documents": 52000,
    "active_documents": 48500,
    "average_price": 325.50,
    "unique_cities": 45,
    "unique_categories": 28
  },
  "version": "0.2.0"
}
```

4 ENDPOINT : /health

Méthode : GET

Rôle : Vérification rapide du statut du service

Exemple de requête

```
bash
```

```
curl http://localhost:8000/health
```

Réponse (service OK)

```
json

{
  "ok": true,
  "indexed": 50000,
  "embedder_ready": true,
  "reranker_ready": true,
  "mongodb_ready": true
}
```

Utilisation

- Monitoring automatique (Kubernetes, Docker)
- Load balancers
- Scripts de surveillance
- Tests d'intégration

⚙️ VARIABLES D'ENVIRONNEMENT

Configuration complète du fichier .env

Section 1 : Chemins des modèles

```
env

# Chemin vers le modèle d'embedding (BGE-M3)
EMBED_MODEL_PATH=./models/embed

# Chemin vers le modèle de reranking (MS-MARCO)
# Laisser vide pour désactiver le reranking
RERANKER_PATH=./models/reranker
```

Explication :

- Ces chemins indiquent où trouver les modèles d'IA
- Les modèles doivent être téléchargés séparément (gros fichiers)
- Si `RERANKER_PATH` est vide, le reranking sera désactivé

Section 2 : Index vectoriel

```
env
```

```
# Dossier où l'index sera sauvegardé
```

```
INDEX_DIR=./indexes
```

Explication :

- L'index contient tous les vecteurs des annonces
- Crée automatiquement au premier démarrage
- Taille approximative : 50 MB par 10 000 annonces

Section 3 : MongoDB

```
env
```

```
# URI de connexion à MongoDB
```

```
MONGODB_URI=mongodb://localhost:27017
```

```
# Nom de la base de données
```

```
MONGO_DB=kira
```

```
# Nom de la collection contenant les annonces
```

```
MONGO_COLL=posts
```

```
# Activer la synchronisation automatique (Change Streams)
```

```
ENABLE_CHANGE_STREAMS=false
```

Explication :

Variable	Exemple	Description
MONGODB_URI	mongodb://localhost:27017	Adresse du serveur MongoDB
	mongodb://user:pass@host:27017	Avec authentification
	mongodb://host1:27017,host2:27017	Cluster MongoDB
MONGO_DB	kira	Nom de votre base de données
MONGO_COLL	posts	Collection contenant les annonces
ENABLE_CHANGE_STREAMS	true/false	Synchronisation temps réel

Change Streams :

- Si true : Le service détecte automatiquement les ajouts/modifications/suppressions
- Si false : Il faut appeler /admin/rebuild manuellement pour synchroniser

Section 4 : Réseau du service

env

Adresse d'écoute du service

SERVICE_HOST=0.0.0.0

Port d'écoute

SERVICE_PORT=8000

Explanation :

Variable	Valeur	Usage
SERVICE_HOST	0.0.0.0	Écoute sur toutes les interfaces (production)
	127.0.0.1	Écoute locale uniquement (développement)
SERVICE_PORT	8000	Port standard (peut être changé)

Section 5 : Paramètres de recherche

env

Nombre de candidats à récupérer lors de la recherche vectorielle

RETRIEVE_K=200

Nombre de résultats finaux à retourner par défaut

FINAL_TOPK=10

Explanation :

RETRIEVE_K : Nombre de documents récupérés avant le reranking

- Plus grand = meilleure qualité mais plus lent
- Recommandé : 100-500
- Notre choix : 200 (bon compromis)

FINAL_TOPK : Nombre de résultats retournés à l'utilisateur

- Généralement 10-20 résultats
- Peut être modifié par requête avec le paramètre topk

Section 6 : Performance

env

```
# Taille des batches pour l'embedding  
EMBEDDING_BATCH_SIZE=32  
  
# Longueur maximale d'une requête (caractères)  
MAX_QUERY_LENGTH=1000  
  
# Nombre maximum de candidats lors du filtrage MongoDB  
MAX_CANDIDATES_FILTER=50000
```

Explication :

Variable	Impact	Recommandation
EMBEDDING_BATCH_SIZE	Mémoire et vitesse	16-64 selon RAM
MAX_QUERY_LENGTH	Protection contre abus	500-2000
MAX_CANDIDATES_FILTER	Limite des filtres MongoDB	20000-100000

Section 7 : Logging

env

```
# Niveau de détail des logs  
LOG_LEVEL=INFO
```

Niveaux disponibles :

Niveau	Détail	Usage
DEBUG	Maximum	Développement, dépannage
INFO	Standard	Production normale
WARNING	Avertissements	Production optimisée
ERROR	Erreurs uniquement	Production minimale

Exemples de configurations

Configuration DÉVELOPPEMENT

env

```
EMBED_MODEL_PATH=./models/embed
RERANKER_PATH=./models/reranker
INDEX_DIR=./indexes

MONGODB_URI=mongodb://localhost:27017
MONGO_DB=kira_dev
MONGO_COLL=posts
ENABLE_CHANGE_STREAMS=true

SERVICE_HOST=127.0.0.1
SERVICE_PORT=8000

RETRIEVE_K=100
FINAL_TOPK=5

EMBEDDING_BATCH_SIZE=16
LOG_LEVEL=DEBUG
```

Configuration PRODUCTION

```
env

EMBED_MODEL_PATH=/opt/kira/models/embed
RERANKER_PATH=/opt/kira/models/reranker
INDEX_DIR=/var/kira/indexes

MONGODB_URI=mongodb://user:pass@mongo-cluster:27017/kira?ssl=true
MONGO_DB=kira_prod
MONGO_COLL=posts
ENABLE_CHANGE_STREAMS=true

SERVICE_HOST=127.0.0.1
SERVICE_PORT=8000

RETRIEVE_K=200
FINAL_TOPK=10

EMBEDDING_BATCH_SIZE=64
MAX_QUERY_LENGTH=500
LOG_LEVEL=WARNING
```

GUIDE DE DÉMARRAGE RAPIDE

Étape 1 : Installation

```
bash

# Cloner le projet
git clone https://github.com/votre-org/kira-search-service
cd kira-search-service

# Créer l'environnement virtuel
python -m venv .venv

# Activer l'environnement
# Sur Linux/Mac :
source .venv/bin/activate
# Sur Windows :
.venv\Scripts\activate

# Installer les dépendances
pip install -r requirements.txt
```

Étape 2 : Configuration

```
bash

# Copier le fichier de configuration
cp .env.example .env

# Éditer avec vos paramètres
nano .env
```

Étape 3 : Télécharger les modèles

```
python
```

```
# Créer un script download_models.py
from sentence_transformers import SentenceTransformer, CrossEncoder

# Télécharger le modèle d'embedding
print("Téléchargement du modèle d'embedding...")
embedder = SentenceTransformer('BAAI/bge-m3')
embedder.save('./models/embed')

# Télécharger le modèle de reranking
print("Téléchargement du modèle de reranking...")
reranker = CrossEncoder('cross-encoder/ms-marco-MiniLM-L-6-v2')
reranker.save('./models/reranker')

print("Téléchargement terminé !")
```

bash

```
# Exécuter le script
python download_models.py
```

Étape 4 : Démarrer le service

bash

```
python run.py
```

Étape 5 : Vérifier le fonctionnement

bash

```
# Test de santé
curl http://localhost:8000/health

# Devrait retourner :
# {"ok": true, "indexed": 0, ...}
```

Étape 6 : Construire l'index

bash

```
# Si vous avez des données dans MongoDB
curl -X POST http://localhost:8000/admin/rebuild

# Sinon, continuez au test de recherche
```

Étape 7 : Test de recherche

```
bash
```

```
curl -X POST http://localhost:8000/search \  
-H "Content-Type: application/json" \  
-d '{"q": "test"}'
```