

Penjelasan Detail Kode Sistem Tiket RAG

1. IMPORT LIBRARIES

```
python

import os
import json
import logging
from datetime import datetime
import psycopg2
from psycopg2.extras import RealDictCursor
from langchain_google_genai import ChatGoogleGenerativeAI, GoogleGenerativeAIEmbeddings
# ... dan library lainnya
```

Fungsi: Mengimpor semua library yang diperlukan untuk:

- **os, json:** Operasi sistem dan manipulasi JSON
- **logging:** Sistem pencatatan log
- **psycopg2:** Koneksi dan operasi database PostgreSQL
- **langchain:** Framework untuk membangun aplikasi LLM
- **pydantic:** Validasi dan parsing data
- **dotenv:** Membaca environment variables

2. KONFIGURASI

```
python

DB_URI = "postgresql://postgres:1234@localhost:5432/postgres"
VECTOR_DB_CONNECTION = "postgresql+psycopg2://langchain:langchain@localhost:6024/langchain"
COLLECTION_NAME = "my_docs"
```

Fungsi: Mendefinisikan konstanta konfigurasi untuk:

- **DB_URI:** String koneksi database utama untuk data tiket
- **VECTOR_DB_CONNECTION:** String koneksi database vektor untuk penyimpanan embeddings
- **COLLECTION_NAME:** Nama koleksi dalam vector database

Logging Configuration

```
python
```

```
logging.basicConfig(  
    level=logging.INFO,  
    format='%(asctime)s - %(levelname)s - %(message)s',  
    handlers=[  
        logging.FileHandler('D:\\interface\\ticketing_system_cron.log'),  
        logging.StreamHandler()  
    ]  
)
```

Fungsi: Mengkonfigurasi sistem logging untuk:

- Mencatat ke file log dan console
- Format timestamp, level, dan pesan
- Level INFO dan di atasnya

3. PYDANTIC MODELS

```
python
```

```
class TicketAnalysis(BaseModel):  
    issue: str = Field(description="Salin ulang pertanyaan/keluhan dari pengguna")  
    priority: str = Field(description="Pilih salah satu: P1 (Kritis), P2 (Tinggi), P3 (Sedang), P4 (Rendah)")  
    unit: str = Field(description="Rekomendasikan tim atau departemen yang paling sesuai")  
    solution: str = Field(description="Solusi awal yang dapat membantu pengguna")  
    justification: str = Field(description="Justifikasi kenapa keluhan tersebut memiliki prioritas tertentu")
```

Fungsi: Mendefinisikan struktur data hasil analisis tiket dengan validasi otomatis. Model ini memastikan output AI memiliki format yang konsisten dengan 5 field wajib.

4. DATABASE FUNCTIONS

get_db_connection()

```
python
```

```
def get_db_connection():
    try:
        connection = psycopg2.connect(
            host=DB_HOST,
            database=DB_NAME,
            user=DB_USER,
            password=DB_PASSWORD,
            port=DB_PORT
        )
        return connection
    except psycopg2.Error as e:
        logger.error(f"Database connection error: {e}")
        return None
```

Fungsi: Membuat koneksi ke database PostgreSQL dengan error handling. Mengembalikan objek koneksi jika berhasil, None jika gagal.

update_ticket_status()

```
python

def update_ticket_status(ticket_id, new_status):
    conn = None
    try:
        conn = get_db_connection()
        if conn is None:
            return False

        with conn.cursor() as cur:
            cur.execute("""
                UPDATE data.rag_queries
                SET status = %s
                WHERE id = %s
            """, (new_status, ticket_id))
            conn.commit()
            logger.info(f"Ticket {ticket_id} status updated to {new_status}")
            return True
    # ... error handling
```

Fungsi: Mengubah status tiket dalam database. Status melacak tahap pemrosesan (pending → processing → done). Menggunakan parameterized query untuk keamanan SQL injection.

save_rag_result()

python

```
def save_rag_result(ticket_id, issue, solution, sources):  
    # ... implementasi serupa dengan update_ticket_status  
    cur.execute("""  
        INSERT INTO data.rag_logs (query_id, issue, solution, sources)  
        VALUES (%s, %s, %s, %s)  
        """, (ticket_id, issue, json.dumps(solution, ensure_ascii=False), json.dumps(sources, ensure_ascii=False)))
```

Fungsi: Menyimpan hasil analisis RAG ke database. Data solution dan sources disimpan dalam format JSON dengan encoding UTF-8.

5. RAG SYSTEM INITIALIZATION

initialize_rag_system()

python

```
def initialize_rag_system():  
    try:  
        chat_model = ChatGoogleGenerativeAI(  
            google_api_key=GOOGLE_API_KEY,  
            model='gemini-2.5-pro',  
            temperature=0.3  
        )  
  
        embeddings = GoogleGenerativeAIEmbeddings(  
            model="models/embedding-001",  
            google_api_key=GOOGLE_API_KEY  
        )
```

Fungsi: Inisialisasi komponen RAG system:

Chat Model

- **Model:** Gemini 2.5 Pro untuk analisis teks
- **Temperature:** 0.3 (output lebih deterministik)

Embeddings Model

- **Model:** Google's embedding-001 untuk konversi teks ke vektor

Database Integration

python

```
db = SQLiteDatabase.from_uri(DB_URI, schema="data")

def load_documents_from_db():
    with db._engine.connect() as conn:
        result = conn.execute(text("""
            SELECT DISTINCT "ID Tiket", "Keluhan", "Prioritas", "Justifikasi Prioritas", "Unit Penanggung Jawab", "Solusi Awal"
            FROM data.dataset_dummy_ticketing
            WHERE "Keluhan" IS NOT NULL AND "Prioritas" IS NOT NULL;
        """))
```

Fungsi: Memuat data historis tiket dari database untuk dijadikan basis pengetahuan. Data ini akan digunakan untuk mencari tiket serupa.

Vector Store Setup

python

```
vector_store = PGVector(
    embeddings=embeddings,
    collection_name=COLLECTION_NAME,
    connection=VECTOR_DB_CONNECTION,
    use_jsonb=True,
)

# Hapus dan buat ulang untuk data terbaru
vector_store.delete_collection()
vector_store.create_collection()

# Tambahkan dokumen dalam batch
batch_size = 100
for i in range(0, len(docs), batch_size):
    batch = docs[i:i + batch_size]
    vector_store.add_documents(batch)
```

Fungsi: Menyiapkan vector database untuk similarity search:

- Menghapus koleksi lama untuk memastikan data fresh
- Menambahkan dokumen dalam batch untuk efisiensi
- Setiap dokumen dikonversi ke embedding vector

Chain Creation

```
python

system_prompt = (
    "Anda adalah asisten AI yang sangat efisien untuk tim dukungan IT..."
    "1. issue: Salin ulang pertanyaan/keluhan dari pengguna.\n"
    "2. priority: Pilih salah satu dari P1 (Kritis), P2 (Tinggi), P3 (Sedang), atau P4 (Rendah).\n"
    # ... instruksi lengkap
)

prompt = ChatPromptTemplate.from_messages([
    ("system", system_prompt),
    ("human", "{input}"),
])

question_answer_chain = create_stuff_documents_chain(chat_model, prompt)
rag_chain = create_retrieval_chain(retriever, question_answer_chain)
```

Fungsi: Membangun chain RAG yang menggabungkan:

- **Retriever:** Mencari dokumen relevan dari vector store
- **LLM Chain:** Menganalisis dengan model AI
- **Parser:** Mengonversi output ke format TicketAnalysis

Output Parser

```
python
```

```

def parse_rag_output(inputs):
    result = None
    try:
        result = rag_chain.invoke(inputs)
        parsed_answer = parser.parse(result['answer'])

        # Validasi field wajib
        required_fields = ['issue', 'priority', 'unit', 'solution', 'justification']
        for field in required_fields:
            if field not in parsed_answer or not parsed_answer[field]:
                parsed_answer[field] = 'Tidak Diketahui'

        # Validasi prioritas
        valid_priorities = ['P1 (Kritis)', 'P2 (Tinggi)', 'P3 (Sedang)', 'P4 (Rendah)']
        if parsed_answer['priority'] not in valid_priorities:
            parsed_answer['priority'] = 'P3 (Sedang)'

```

Fungsi: Memproses output RAG dengan:

- Parsing JSON output dari AI
- Validasi semua field wajib ada
- Fallback ke nilai default jika ada error
- Validasi prioritas sesuai dengan list yang valid

6. PROCESSING FUNCTIONS

process_single_ticket()

python

```
def process_single_ticket(ticket_id, question, rag_chain_parser):
    try:
        logger.info(f"Processing ticket {ticket_id}: {question[:50]}...")

        # Update status ke 'processing'
        if not update_ticket_status(ticket_id, 'processing'):
            return False, "Failed to update ticket status to 'processing'"

        # Jalankan RAG analysis
        result = rag_chain_parser({"input": question})
        parsed_answer = result['answer']

        # Simpan hasil
        sources = [{"content": doc.page_content, "metadata": doc.metadata} for doc in result['context']]

        if save_rag_result(ticket_id, question, parsed_answer, sources):
            if update_ticket_status(ticket_id, 'done'):
                return True, parsed_answer
```

Fungsi: Memproses satu tiket tunggal dengan langkah:

1. Update status ke 'processing'
2. Jalankan analisis RAG
3. Extract sources yang digunakan
4. Simpan hasil ke database
5. Update status ke 'done'
6. Handle error dengan rollback status

process_tickets_batch()

python


```
def process_tickets_batch(rag_chain_parser, max_tickets=5):
    try:
        conn = get_db_connection()
        with conn.cursor(cursor_factory=RealDictCursor) as cur:
            cur.execute("""
                SELECT id, question FROM data.rag_queries
                WHERE status = 'pending' ORDER BY id ASC LIMIT %s
            """, (max_tickets,))
            pending_tickets = cur.fetchall()

        results = []
        for ticket in pending_tickets:
            success, result_data = process_single_ticket(
                ticket['id'], ticket['question'], rag_chain_parser
            )
            results.append({
                'ticket_id': ticket['id'],
                'success': success,
                'result': result_data
            })
```

Fungsi: Memproses batch tiket dengan:

- Query tiket dengan status 'pending'
- Limit jumlah tiket yang diproses per batch
- Proses setiap tiket secara berurutan
- Kumpulkan hasil dalam list untuk reporting
- Rate limiting untuk menghindari API limits

7. MAIN EXECUTION

run_job()

```
python
```

```
def run_job():
    logger.info("Cron job started: Processing pending tickets.")

    # Inisialisasi RAG system
    rag_chain_parser, db = initialize_rag_system()
    if not rag_chain_parser:
        logger.error("Failed to initialize RAG system. Aborting job.")
        return

    try:
        max_tickets_to_process = 10
        result = process_tickets_batch(rag_chain_parser, max_tickets=max_tickets_to_process)

        # Log hasil batch processing
        if 'results' in result:
            processed_count = len(result['results'])
            success_count = sum(1 for r in result['results'] if r['success'])
            logger.info(f"Batch processing finished. Processed: {processed_count}, Successful: {success_count}.")
```

Fungsi: Entry point utama yang:

- Inisialisasi sistem RAG
- Proses batch tiket (max 10)
- Log statistik hasil pemrosesan
- Handle error global dengan logging

Entry Point

```
python

if __name__ == "__main__":
    run_job()
```

Fungsi: Menjalankan job ketika script dieksekusi langsung, biasanya oleh cron scheduler.

ALUR KERJA SISTEM

1. **Inisialisasi:** Load model AI, setup vector database, buat knowledge base dari data historis
2. **Query Tiket:** Ambil tiket dengan status 'pending' dari database
3. **Similarity Search:** Cari tiket historis yang mirip menggunakan vector search
4. **AI Analysis:** Gunakan Gemini untuk menganalisis tiket baru berdasarkan konteks historis

5. **Validation:** Validasi dan format output sesuai TicketAnalysis model
6. **Save Results:** Simpan hasil analisis dan source documents ke database
7. **Status Update:** Update status tiket ke 'done'
8. **Logging:** Catat semua aktivitas untuk monitoring

KEAMANAN & BEST PRACTICES

- **Parameterized Queries:** Mencegah SQL injection
- **Error Handling:** Comprehensive error handling dengan rollback
- **Connection Management:** Proper database connection management
- **Rate Limiting:** Jeda antar request untuk menghindari API limits
- **Logging:** Detailed logging untuk troubleshooting
- **Batch Processing:** Efisien untuk volume besar
- **Data Validation:** Strict validation menggunakan Pydantic