"""

Logging configuration for the RAG pipeline.

This module sets up structured logging with both file and console handlers,

including log rotation and formatting suitable for production environments.

"""

import logging

import logging.handlers

import sys

from pathlib import Path

from typing import Optional

import json

from datetime import datetime

class StructuredFormatter(logging.Formatter):

"""Custom formatter that outputs structured JSON logs."""

def format(self, record: logging.LogRecord) -> str:

"""Format log record as JSON."""

log\_obj = {

"timestamp": datetime.utcnow().isoformat(),

"level": record.levelname,

"logger": record.name,

"message": record.getMessage(),

"module": record.module,

"function": record.funcName,

"line": record.lineno,

}

# Add exception info if present

if record.exc\_info:

log\_obj["exception"] = self.formatException(record.exc\_info)

# Add extra fields

for key, value in record.\_\_dict\_\_.items():

if key not in ["name", "msg", "args", "created", "filename", "funcName",

"levelname", "levelno", "lineno", "module", "msecs",

"pathname", "process", "processName", "relativeCreated",

"thread", "threadName", "exc\_info", "exc\_text", "stack\_info"]:

log\_obj[key] = value

return json.dumps(log\_obj)

def setup\_logging(

log\_level: str = "INFO",

log\_file: Optional[str] = None,

max\_bytes: int = 10485760, # 10MB

backup\_count: int = 5,

enable\_console: bool = True

) -> None:

"""

Configure logging for the application.

Args:

log\_level: Logging level (DEBUG, INFO, WARNING, ERROR, CRITICAL)

log\_file: Path to log file. If None, only console logging is enabled

max\_bytes: Maximum size of log file before rotation

backup\_count: Number of backup files to keep

enable\_console: Whether to enable console logging

"""

# Create logs directory if needed

if log\_file:

log\_path = Path(log\_file)

log\_path.parent.mkdir(parents=True, exist\_ok=True)

# Configure root logger

root\_logger = logging.getLogger()

root\_logger.setLevel(getattr(logging, log\_level.upper()))

# Remove existing handlers

root\_logger.handlers = []

# Console handler with human-readable format

if enable\_console:

console\_handler = logging.StreamHandler(sys.stdout)

console\_handler.setLevel(getattr(logging, log\_level.upper()))

console\_formatter = logging.Formatter(

'%(asctime)s - %(name)s - %(levelname)s - %(message)s',

datefmt='%Y-%m-%d %H:%M:%S'

)

console\_handler.setFormatter(console\_formatter)

root\_logger.addHandler(console\_handler)

# File handler with JSON format for production

if log\_file:

file\_handler = logging.handlers.RotatingFileHandler(

log\_file,

maxBytes=max\_bytes,

backupCount=backup\_count

)

file\_handler.setLevel(getattr(logging, log\_level.upper()))

file\_handler.setFormatter(StructuredFormatter())

root\_logger.addHandler(file\_handler)

# Configure third-party loggers

logging.getLogger("urllib3").setLevel(logging.WARNING)

logging.getLogger("httpx").setLevel(logging.WARNING)

logging.getLogger("openai").setLevel(logging.INFO)

logging.getLogger("langchain").setLevel(logging.INFO)

logging.getLogger("chromadb").setLevel(logging.WARNING)

def get\_logger(name: str) -> logging.Logger:

"""

Get a logger instance with the given name.

Args:

name: Logger name (typically \_\_name\_\_)

Returns:

Configured logger instance

"""

return logging.getLogger(name)

class LogContext:

"""Context manager for adding context to log messages."""

def \_\_init\_\_(self, logger: logging.Logger, \*\*kwargs):

"""Initialize with logger and context fields."""

self.logger = logger

self.context = kwargs

self.old\_factory = None

def \_\_enter\_\_(self):

"""Enter context and add fields to log records."""

old\_factory = logging.getLogRecordFactory()

self.old\_factory = old\_factory

def record\_factory(\*args, \*\*kwargs):

record = old\_factory(\*args, \*\*kwargs)

for key, value in self.context.items():

setattr(record, key, value)

return record

logging.setLogRecordFactory(record\_factory)

return self

def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):

"""Exit context and restore original factory."""

if self.old\_factory:

logging.setLogRecordFactory(self.old\_factory)