**1. Thêm Dependencies**

**pom.xml:**

xml

<dependencies>

*<!-- Redis dependencies -->*

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis</artifactId>

</dependency>

*<!-- JSON processing -->*

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

</dependencies>

**2. Job Model**

java

package com.jpd.model;

import java.io.Serializable;

import java.time.LocalDateTime;

public class WhisperJob implements Serializable {

private String jobId;

private byte[] audioData;

private String audioFilename;

private String expectedText;

private LocalDateTime createdAt;

private String status; *// QUEUED, PROCESSING, COMPLETED, FAILED*

*// Constructors*

public WhisperJob() {}

public WhisperJob(String jobId, byte[] audioData, String audioFilename, String expectedText) {

this.jobId = jobId;

this.audioData = audioData;

this.audioFilename = audioFilename;

this.expectedText = expectedText;

this.createdAt = LocalDateTime.now();

this.status = "QUEUED";

}

*// Getters and Setters*

*// ... (tạo đầy đủ getters/setters)*

}

**3. Redis Configuration**

java

package com.jpd.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.redis.connection.RedisConnectionFactory;

import org.springframework.data.redis.core.RedisTemplate;

import org.springframework.data.redis.serializer.GenericJackson2JsonRedisSerializer;

import org.springframework.data.redis.serializer.StringRedisSerializer;

@Configuration

public class RedisConfig {

@Bean

public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory connectionFactory) {

RedisTemplate<String, Object> template = new RedisTemplate<>();

template.setConnectionFactory(connectionFactory);

*// Sử dụng String serializer cho key*

template.setKeySerializer(new StringRedisSerializer());

template.setHashKeySerializer(new StringRedisSerializer());

*// Sử dụng JSON serializer cho value*

template.setValueSerializer(new GenericJackson2JsonRedisSerializer());

template.setHashValueSerializer(new GenericJackson2JsonRedisSerializer());

return template;

}

}

**4. Queue Service**

java

package com.jpd.service;

import java.util.Map;

import java.util.UUID;

import java.util.concurrent.TimeUnit;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.redis.core.RedisTemplate;

import org.springframework.stereotype.Service;

import org.springframework.web.multipart.MultipartFile;

import com.jpd.model.WhisperJob;

@Service

public class QueueService {

@Autowired

private RedisTemplate<String, Object> redisTemplate;

private static final String QUEUE\_KEY = "whisper\_queue";

private static final String RESULT\_PREFIX = "result:";

private static final String JOB\_PREFIX = "job:";

*/\*\**

\* Đưa job vào queue

*\*/*

public String submitJob(MultipartFile audioFile, String expectedText) throws Exception {

try {

String jobId = UUID.randomUUID().toString();

*// Tạo job object*

WhisperJob job = new WhisperJob(

jobId,

audioFile.getBytes(),

audioFile.getOriginalFilename(),

expectedText

);

*// Lưu job info*

redisTemplate.opsForValue().set(JOB\_PREFIX + jobId, job, 1, TimeUnit.HOURS);

*// Đưa job ID vào queue*

redisTemplate.opsForList().leftPush(QUEUE\_KEY, jobId);

return jobId;

} catch (Exception e) {

throw new Exception("Failed to submit job: " + e.getMessage(), e);

}

}

*/\*\**

\* Lấy kết quả job

*\*/*

@SuppressWarnings("unchecked")

public Map<String, Object> getJobResult(String jobId) {

*// Kiểm tra kết quả*

Object result = redisTemplate.opsForValue().get(RESULT\_PREFIX + jobId);

if (result != null) {

return (Map<String, Object>) result;

}

*// Kiểm tra job còn tồn tại không*

WhisperJob job = (WhisperJob) redisTemplate.opsForValue().get(JOB\_PREFIX + jobId);

if (job == null) {

return Map.of("status", "NOT\_FOUND", "message", "Job not found");

}

return Map.of("status", "PROCESSING", "message", "Job is still being processed");

}

*/\*\**

\* Lấy status của job

*\*/*

public String getJobStatus(String jobId) {

*// Check kết quả trước*

if (redisTemplate.hasKey(RESULT\_PREFIX + jobId)) {

return "COMPLETED";

}

*// Check job có tồn tại*

if (redisTemplate.hasKey(JOB\_PREFIX + jobId)) {

return "PROCESSING";

}

return "NOT\_FOUND";

}

}

**5. Refactor AudioService**

java

package com.jpd.service;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.web.multipart.MultipartFile;

@Service

public class AudioService {

@Autowired

private QueueService queueService;

*/\*\**

\* Submit job async - trả về jobId ngay lập tức

*\*/*

public String submitEvaluationJob(MultipartFile file, String expectedAnswer) throws Exception {

return queueService.submitJob(file, expectedAnswer);

}

*/\*\**

\* Lấy kết quả job

*\*/*

public Map<String, Object> getEvaluationResult(String jobId) {

return queueService.getJobResult(jobId);

}

*/\*\**

\* Check status job

*\*/*

public String getJobStatus(String jobId) {

return queueService.getJobStatus(jobId);

}

}

**6. Refactor Controller**

java

package com.jpd.controller;

import java.util.Map;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import org.springframework.web.multipart.MultipartFile;

import com.jpd.service.AudioService;

@CrossOrigin(origins = "http://localhost:3000", allowedHeaders = "\*",

methods = {RequestMethod.POST, RequestMethod.GET})

@RestController

@RequestMapping("/api")

public class ListeningPageController {

private final AudioService audioService;

public ListeningPageController(AudioService audioService) {

this.audioService = audioService;

}

*/\*\**

\* Submit job - trả về jobId ngay lập tức

*\*/*

@PostMapping("/evaluate")

public ResponseEntity<?> submitEvaluation(@RequestParam("audio") MultipartFile file,

@RequestParam("sentence") String expectedAnswer) {

try {

String jobId = audioService.submitEvaluationJob(file, expectedAnswer);

return ResponseEntity.ok(Map.of(

"jobId", jobId,

"status", "QUEUED",

"message", "Job submitted successfully"

));

} catch (Exception e) {

return ResponseEntity.status(500).body(Map.of("error", e.getMessage()));

}

}

*/\*\**

\* Lấy kết quả job

*\*/*

@GetMapping("/evaluate/{jobId}")

public ResponseEntity<?> getEvaluationResult(@PathVariable String jobId) {

try {

Map<String, Object> result = audioService.getEvaluationResult(jobId);

return ResponseEntity.ok(result);

} catch (Exception e) {

return ResponseEntity.status(500).body(Map.of("error", e.getMessage()));

}

}

*/\*\**

\* Check status job (optional)

*\*/*

@GetMapping("/evaluate/{jobId}/status")

public ResponseEntity<?> getJobStatus(@PathVariable String jobId) {

String status = audioService.getJobStatus(jobId);

return ResponseEntity.ok(Map.of("status", status));

}

}

**7. Frontend Usage (React)**

javascript

*// Submit job*

const submitAudio = async (audioFile, expectedText) => {

const formData = new FormData();

formData.append('audio', audioFile);

formData.append('sentence', expectedText);

const response = await fetch('/api/evaluate', {

method: 'POST',

body: formData

});

const result = await response.json();

return result.jobId; *// Lấy jobId*

};

*// Polling kết quả*

const pollResult = async (jobId) => {

const checkResult = async () => {

const response = await fetch(`/api/evaluate/${jobId}`);

const result = await response.json();

if (result.status === 'COMPLETED') {

return result;

} else if (result.status === 'PROCESSING') {

*// Đợi 2 giây rồi check lại*

setTimeout(checkResult, 2000);

}

};

return checkResult();

};

Bạn có muốn tôi tiếp tục với phần Python Worker để xử lý jobs từ queue không?

Réessayer

Claude n'a pas encore la capacité d'exécuter le code qu'il génère.

Á

như này chưa đủ để xử lí job từ queue ak

Modifier

Đúng rồi!

Réessayer

# whisper\_service.py - Sử dụng numpy array thay vì file from fastapi import FastAPI, Form, File, UploadFile, HTTPException import whisper import difflib import numpy as np import librosa from sentence\_transformers import SentenceTransformer, util import io app = FastAPI() # Tải mô hình

pasted

Á

như này chưa đủ để xử lí job từ queue ak Modifi

Modifier

Đúng rồi! Chưa đủ. Tôi thiếu phần **Python Worker** để xử lý jobs từ queue. Đây là phần còn thiếu:

**Python Worker - Xử lý jobs từ Redis Queue**

**1. Cài đặt thêm dependencies**

bash

pip install redis celery

**2. Tạo Worker để xử lý queue**

**whisper\_worker.py:**

python

import redis

import json

import base64

import io

import numpy as np

import librosa

import whisper

import difflib

from sentence\_transformers import SentenceTransformer, util

import time

import logging

*# Setup logging*

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

class WhisperWorker:

def \_\_init\_\_(self):

*# Kết nối Redis*

self.redis\_client = redis.Redis(host='localhost', port=6379, db=0)

*# Load models một lần*

logger.info("Loading Whisper model...")

self.whisper\_model = whisper.load\_model("small")

logger.info("Whisper model loaded!")

logger.info("Loading semantic model...")

self.semantic\_model = SentenceTransformer('sentence-transformers/LaBSE')

logger.info("Semantic model loaded!")

*# Queue keys*

self.queue\_key = "whisper\_queue"

self.job\_prefix = "job:"

self.result\_prefix = "result:"

def calc\_similarity(self, ans, pred):

"""Tính độ tương đồng bằng SequenceMatcher"""

return round(difflib.SequenceMatcher(None, ans, pred).ratio() \* 100, 2)

def is\_match(self, user\_text, reference\_text, threshold=0.2):

"""Tính độ tương đồng semantic giữa hai văn bản"""

try:

emb1 = self.semantic\_model.encode(user\_text, convert\_to\_tensor=True)

emb2 = self.semantic\_model.encode(reference\_text, convert\_to\_tensor=True)

similarity = util.cos\_sim(emb1, emb2).item()

return {

"match": similarity >= threshold,

"similarity\_score": round(similarity, 4),

"user\_answer": user\_text,

"expected\_answer": reference\_text,

"status": "COMPLETED"

}

except Exception as e:

logger.error(f"Error in semantic matching: {str(e)}")

return {

"status": "FAILED",

"error": f"Semantic matching error: {str(e)}"

}

def audio\_bytes\_to\_numpy(self, audio\_bytes):

"""Chuyển audio bytes thành numpy array"""

try:

audio\_array, sample\_rate = librosa.load(

io.BytesIO(audio\_bytes),

sr=16000, *# Whisper yêu cầu sample rate 16kHz*

mono=True *# Chuyển về mono*

)

return audio\_array

except Exception as e:

raise Exception(f"Error converting audio to numpy: {str(e)}")

def process\_job(self, job\_data):

"""Xử lý một job"""

try:

job\_id = job\_data['jobId']

audio\_b64 = job\_data['audioData']

expected\_text = job\_data['expectedText']

logger.info(f"Processing job {job\_id}")

*# Decode audio từ base64*

audio\_bytes = base64.b64decode(audio\_b64)

*# Chuyển thành numpy array*

audio\_array = self.audio\_bytes\_to\_numpy(audio\_bytes)

*# Transcribe audio*

result = self.whisper\_model.transcribe(audio\_array, language="ja")

recognized\_text = result["text"].strip()

*# Đánh giá semantic similarity*

evaluation\_result = self.is\_match(recognized\_text, expected\_text)

*# Lưu kết quả vào Redis*

self.redis\_client.setex(

f"{self.result\_prefix}{job\_id}",

3600, *# TTL 1 hour*

json.dumps(evaluation\_result)

)

*# Xóa job data*

self.redis\_client.delete(f"{self.job\_prefix}{job\_id}")

logger.info(f"Job {job\_id} completed successfully")

return True

except Exception as e:

logger.error(f"Error processing job {job\_data.get('jobId', 'unknown')}: {str(e)}")

*# Lưu lỗi vào Redis*

error\_result = {

"status": "FAILED",

"error": str(e)

}

if 'jobId' in job\_data:

self.redis\_client.setex(

f"{self.result\_prefix}{job\_data['jobId']}",

3600,

json.dumps(error\_result)

)

return False

def run(self):

"""Chạy worker liên tục"""

logger.info("Worker started, waiting for jobs...")

while True:

try:

*# Lấy job từ queue (blocking với timeout 5s)*

job\_data = self.redis\_client.brpop(self.queue\_key, timeout=5)

if job\_data:

*# job\_data là tuple (queue\_name, job\_id)*

job\_id = job\_data[1].decode('utf-8')

*# Lấy job detail từ Redis*

job\_detail = self.redis\_client.get(f"{self.job\_prefix}{job\_id}")

if job\_detail:

job\_json = json.loads(job\_detail.decode('utf-8'))

self.process\_job(job\_json)

else:

logger.warning(f"Job detail not found for job\_id: {job\_id}")

except KeyboardInterrupt:

logger.info("Worker stopped by user")

break

except Exception as e:

logger.error(f"Worker error: {str(e)}")

time.sleep(1) *# Đợi 1s trước khi thử lại*

if \_\_name\_\_ == "\_\_main\_\_":

worker = WhisperWorker()

worker.run()

**3. Cập nhật Spring QueueService để phù hợp**

**QueueService.java:**

java

package com.jpd.service;

import java.util.HashMap;

import java.util.Map;

import java.util.UUID;

import java.util.concurrent.TimeUnit;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.redis.core.RedisTemplate;

import org.springframework.stereotype.Service;

import org.springframework.web.multipart.MultipartFile;

import com.fasterxml.jackson.databind.ObjectMapper;

import java.util.Base64;

@Service

public class QueueService {

@Autowired

private RedisTemplate<String, Object> redisTemplate;

private ObjectMapper objectMapper = new ObjectMapper();

private static final String QUEUE\_KEY = "whisper\_queue";

private static final String RESULT\_PREFIX = "result:";

private static final String JOB\_PREFIX = "job:";

*/\*\**

\* Đưa job vào queue

*\*/*

public String submitJob(MultipartFile audioFile, String expectedText) throws Exception {

try {

String jobId = UUID.randomUUID().toString();

*// Encode audio file thành base64*

String audioB64 = Base64.getEncoder().encodeToString(audioFile.getBytes());

*// Tạo job data*

Map<String, Object> jobData = new HashMap<>();

jobData.put("jobId", jobId);

jobData.put("audioData", audioB64);

jobData.put("audioFilename", audioFile.getOriginalFilename());

jobData.put("expectedText", expectedText);

jobData.put("createdAt", System.currentTimeMillis());

*// Lưu job detail*

String jobJson = objectMapper.writeValueAsString(jobData);

redisTemplate.opsForValue().set(JOB\_PREFIX + jobId, jobJson, 1, TimeUnit.HOURS);

*// Đưa job ID vào queue*

redisTemplate.opsForList().leftPush(QUEUE\_KEY, jobId);

return jobId;

} catch (Exception e) {

throw new Exception("Failed to submit job: " + e.getMessage(), e);

}

}

*/\*\**

\* Lấy kết quả job

*\*/*

@SuppressWarnings("unchecked")

public Map<String, Object> getJobResult(String jobId) {

try {

*// Kiểm tra kết quả*

String resultJson = (String) redisTemplate.opsForValue().get(RESULT\_PREFIX + jobId);

if (resultJson != null) {

return objectMapper.readValue(resultJson, Map.class);

}

*// Kiểm tra job còn tồn tại không*

String jobJson = (String) redisTemplate.opsForValue().get(JOB\_PREFIX + jobId);

if (jobJson == null) {

return Map.of("status", "NOT\_FOUND", "message", "Job not found");

}

return Map.of("status", "PROCESSING", "message", "Job is still being processed");

} catch (Exception e) {

return Map.of("status", "ERROR", "message", "Error retrieving job result: " + e.getMessage());

}

}

*/\*\**

\* Lấy status của job

*\*/*

public String getJobStatus(String jobId) {

*// Check kết quả trước*

if (redisTemplate.hasKey(RESULT\_PREFIX + jobId)) {

return "COMPLETED";

}

*// Check job có tồn tại*

if (redisTemplate.hasKey(JOB\_PREFIX + jobId)) {

return "PROCESSING";

}

return "NOT\_FOUND";

}

}

**4. Chạy hệ thống**

**Bước 1:** Khởi động Redis

bash

redis-server

**Bước 2:** Chạy Python Worker

bash

python whisper\_worker.py

**Bước 3:** Chạy Spring Boot application

**Bước 4:** Test với React frontend

**5. Script để chạy nhiều workers**

**start\_workers.py:**

python

import subprocess

import sys

def start\_workers(num\_workers=3):

processes = []

for i in range(num\_workers):

print(f"Starting worker {i+1}...")

process = subprocess.Popen([sys.executable, "whisper\_worker.py"])

processes.append(process)

try:

*# Đợi tất cả processes*

for process in processes:

process.wait()

except KeyboardInterrupt:

print("Stopping all workers...")

for process in processes:

process.terminate()

if \_\_name\_\_ == "\_\_main\_\_":

start\_workers(3) *# Chạy 3 workers*

**Luồng hoạt động hoàn chỉnh:**

1. **React** → gửi audio tới Spring /api/evaluate
2. **Spring** → tạo job, đưa vào Redis queue, trả jobId
3. **Python Worker** → lấy job từ queue, xử lý Whisper, lưu kết quả
4. **React** → polling Spring /api/evaluate/{jobId} để lấy kết quả

Bây giờ hệ thống đã hoàn chỉnh để xử lý queue rồi!

bash

npm install @tanstack/react-query

**2. Setup QueryClient**

javascript

*// main.jsx hoặc App.jsx*

import { QueryClient, QueryClientProvider } from '@tanstack/react-query';

const queryClient = new QueryClient({

defaultOptions: {

queries: {

retry: 3,

refetchOnWindowFocus: false,

},

},

});

function App() {

return (

<QueryClientProvider client={queryClient}>

{*/\* Your app components \*/*}

</QueryClientProvider>

);

}

**3. Hook tối ưu**

javascript

*// hooks/useAsyncEvaluation.js*

import { useMutation, useQuery, useQueryClient } from '@tanstack/react-query';

import { evaluateAnswerSpeaking, getEvaluationResult } from '../api/audioAPI';

export const useAsyncEvaluation = () => {

const queryClient = useQueryClient();

*// Submit job mutation*

const submitMutation = useMutation({

mutationFn: evaluateAnswerSpeaking,

onSuccess: (data) => {

*// Invalidate và refetch result query*

queryClient.invalidateQueries(['evaluation', data.data.jobId]);

}

});

const jobId = submitMutation.data?.data?.jobId;

*// Result query với smart polling*

const resultQuery = useQuery({

queryKey: ['evaluation', jobId],

queryFn: () => getEvaluationResult(jobId),

enabled: !!jobId,

refetchInterval: (data, query) => {

const status = data?.data?.status;

*// Dừng polling nếu hoàn thành*

if (status === 'COMPLETED' || status === 'FAILED' || status === 'NOT\_FOUND') {

return false;

}

*// Polling với backoff strategy*

const failureCount = query.state.failureCount;

return Math.min(2000 \* Math.pow(2, failureCount), 10000); *// 2s, 4s, 8s, max 10s*

},

retry: (failureCount, error) => {

*// Retry tối đa 5 lần*

return failureCount < 5;

}

});

return {

*// Submit functions*

submitEvaluation: submitMutation.mutate,

isSubmitting: submitMutation.isPending,

*// Result data*

result: resultQuery.data?.data,

isLoading: resultQuery.isFetching && !resultQuery.data,

isPolling: resultQuery.isFetching && !!resultQuery.data,

*// Errors*

error: submitMutation.error || resultQuery.error,

*// Metadata*

jobId,

*// Utils*

reset: () => {

submitMutation.reset();

if (jobId) {

queryClient.removeQueries(['evaluation', jobId]);

}

}

};

};

**4. Component sử dụng**

javascript

*// components/SpeakingEvaluation.jsx*

import React, { useState } from 'react';

import { useAsyncEvaluation } from '../hooks/useAsyncEvaluation';

const SpeakingEvaluation = () => {

const [audioFile, setAudioFile] = useState(null);

const [expectedText, setExpectedText] = useState('');

const {

submitEvaluation,

isSubmitting,

result,

isLoading,

isPolling,

error,

jobId,

reset

} = useAsyncEvaluation();

const handleSubmit = (e) => {

e.preventDefault();

if (!audioFile || !expectedText) {

alert('Please provide both audio file and expected text');

return;

}

const formData = new FormData();

formData.append('audio', audioFile);

formData.append('sentence', expectedText);

submitEvaluation(formData);

};

const handleReset = () => {

reset();

setAudioFile(null);

setExpectedText('');

};

return (

<div className="speaking-evaluation">

<form onSubmit={handleSubmit}>

<div>

<input

type="file"

accept="audio/\*"

onChange={(e) => setAudioFile(e.target.files[0])}

disabled={isSubmitting || isLoading}

/>

</div>

<div>

<input

type="text"

value={expectedText}

onChange={(e) => setExpectedText(e.target.value)}

placeholder="Expected text..."

disabled={isSubmitting || isLoading}

/>

</div>

<button type="submit" disabled={isSubmitting || isLoading}>

{isSubmitting ? 'Submitting...' : 'Evaluate'}

</button>

<button type="button" onClick={handleReset}>

Reset

</button>

</form>

{*/\* Status indicators \*/*}

{isSubmitting && <div className="status">Submitting job...</div>}

{isLoading && (

<div className="status">

Processing audio... {jobId && `(Job: ${jobId.slice(0, 8)}...)`}

</div>

)}

{isPolling && <div className="status">⟳ Getting results...</div>}

{*/\* Error \*/*}

{error && (

<div className="error">

{error.response?.data?.error || error.message || 'Something went wrong'}

</div>

)}

{*/\* Results \*/*}

{result && result.status === 'COMPLETED' && (

<div className="result">

<h3>Evaluation Result:</h3>

<div className={`match ${result.match ? 'success' : 'fail'}`}>

<p><strong>Match:</strong> {result.match ? '✅ Yes' : '❌ No'}</p>

<p><strong>Score:</strong> {result.similarity\_score}</p>

<p><strong>Your Answer:</strong> {result.user\_answer}</p>

<p><strong>Expected:</strong> {result.expected\_answer}</p>

</div>

</div>

)}

</div>

);

};

export default SpeakingEvaluation;