

```
===== app.py =====
import os
import uuid
import threading
import time
import io
import json
import csv
import shutil
import subprocess
from datetime import datetime, timedelta
import stripe
from flask import Flask, request, jsonify, render_template, send_file
from google import genai
from google.genai import types
from dotenv import load_dotenv
from werkzeug.utils import secure_filename
from docx import Document
from docx.shared import Pt, Inches
from docx.enum.text import WD_ALIGN_PARAGRAPH
import firebase_admin
from firebase_admin import credentials, auth, firestore

load_dotenv()
app = Flask(__name__)

UPLOAD_FOLDER = 'uploads'
ALLOWED_PDF_EXTENSIONS = {'pdf'}
ALLOWED_AUDIO_EXTENSIONS = {'mp3', 'm4a', 'wav', 'aac', 'ogg', 'flac'}
MAX_CONTENT_LENGTH = 500 * 1024 * 1024

os.makedirs(UPLOAD_FOLDER, exist_ok=True)
client = genai.Client(api_key=os.getenv('GEMINI_API_KEY'))

# --- Firebase Setup ---
if os.path.exists('firebase-credentials.json'):
    cred = credentials.Certificate('firebase-credentials.json')
else:
    firebase_creds = json.loads(os.getenv('FIREBASE_CREDENTIALS', '{}'))
    cred = credentials.Certificate(firebase_creds)

firebase_admin.initialize_app(cred)
db = firestore.client()

# --- Stripe Setup ---
stripe.api_key = os.getenv('STRIPE_SECRET_KEY')
STRIPE_PUBLISHABLE_KEY = os.getenv('STRIPE_PUBLISHABLE_KEY', '')
STRIPE_WEBHOOK_SECRET = os.getenv('STRIPE_WEBHOOK_SECRET', '')
ADMIN_EMAILS = {email.strip().lower() for email in os.getenv('ADMIN_EMAILS', '').split(',') if email.strip()}
ADMIN_UIDS = {uid.strip() for uid in os.getenv('ADMIN_UIDS', '').split(',') if uid.strip()}

# --- In-Memory Storage (jobs only - credits are in Firestore now) ---
jobs = {}

# --- Credit Bundles (what users can buy) ---
CREDIT_BUNDLES = {
    'lecture_5': {
        'name': 'Lecture Notes - 5 Pack',
        'description': '5 standard lecture credits',
        'credits': {'lecture_credits_standard': 5},
    }
}
```

```

    'price_cents': 999,
    'currency': 'eur',
},
'lecture_10': {
    'name': 'Lecture Notes - 10 Pack',
    'description': '10 standard lecture credits (best value)',
    'credits': {'lecture_credits_standard': 10},
    'price_cents': 1699,
    'currency': 'eur',
},
'slides_10': {
    'name': 'Slides Extraction - 10 Pack',
    'description': '10 slides extraction credits',
    'credits': {'slides_credits': 10},
    'price_cents': 499,
    'currency': 'eur',
},
'slides_25': {
    'name': 'Slides Extraction - 25 Pack',
    'description': '25 slides extraction credits (best value)',
    'credits': {'slides_credits': 25},
    'price_cents': 999,
    'currency': 'eur',
},
'interview_3': {
    'name': 'Interview Transcription - 3 Pack',
    'description': '3 interview transcription credits',
    'credits': {'interview_credits_short': 3},
    'price_cents': 799,
    'currency': 'eur',
},
'interview_8': {
    'name': 'Interview Transcription - 8 Pack',
    'description': '8 interview transcription credits (best value)',
    'credits': {'interview_credits_short': 8},
    'price_cents': 1799,
    'currency': 'eur',
},
}

```

# --- Email Allowlist ---

```

ALLOWED_EMAIL_DOMAINS = {
    'gmail.com', 'googlemail.com', 'outlook.com', 'hotmail.com', 'live.com', 'icloud.com', 'yahoo.com', 'yahoo.nl',
    'student.tudelft.nl', 'tudelft.nl', 'uva.nl', 'student.uva.nl', 'vu.nl', 'student.vu.nl',
    'eur.nl', 'student.eur.nl', 'uu.nl', 'students.uu.nl', 'ru.nl', 'student.ru.nl', 'utwente.nl', 'student.utwente.nl',
    'tue.nl', 'student.tue.nl', 'maastrichtuniversity.nl', 'student.maastrichtuniversity.nl', 'leidenuniv.nl',
    'student.leidenuniv.nl', 'rug.nl',
    'student.rug.nl', 'tilburguniversity.edu', 'uvt.nl', 'han.nl', 'student.han.nl',
    'hva.nl', 'student.hva.nl', 'hr.nl',
    'student.hr.nl', 'fontys.nl', 'student.fontys.nl', 'saxion.nl', 'student.saxion.nl',
    'avans.nl', 'student.avans.nl',
    'inholland.nl', 'student.inholland.nl', 'hanze.nl', 'student.hanze.nl', 'zuyd.nl',
    'student.zuyd.nl', 'hu.nl',
    'student.hu.nl', 'windesheim.nl', 'student.windesheim.nl', 'nhlstenden.com',
    'student.nhlstenden.com',
}

```

```

ALLOWED_EMAIL_PATTERNS = ['.edu', '.ac.uk', '.edu.nl']

```

```

def is_email_allowed(email):
    if not email: return False
    email = email.lower()
    domain = email.split('@')[-1] if '@' in email else ''
    if domain in ALLOWED_EMAIL_DOMAINS: return True
    for pattern in ALLOWED_EMAIL_PATTERNS:
        if domain.endswith(pattern): return True
    return False

# --- AI Model Config ---
MODEL_SLIDES = 'gemini-2.5-flash-lite'
MODEL_AUDIO = 'gemini-3-flash-preview'
MODEL_INTEGRATION = 'gemini-2.5-pro'
MODEL_INTERVIEW = 'gemini-2.5-pro'
MODEL_STUDY = 'gemini-2.5-flash-lite'

FREE_LECTURE_CREDITS = 1
FREE_SLIDES_CREDITS = 2
FREE_INTERVIEW_CREDITS = 0

OUTPUT_LANGUAGE_MAP = {
    'dutch': 'Dutch',
    'english': 'English',
    'spanish': 'Spanish',
    'french': 'French',
    'german': 'German',
    'chinese': 'Chinese',
}

# --- Prompts ---
PROMPT_SLIDE_EXTRACTION = """Extraheer alle tekst van de slides uit het bijgevoegde PDF-bestand en identificeer de functie van visuele elementen.
Instructies:
1. Geef per slide duidelijk aan welk slide-nummer het betreft (bv. "Slide 1:").
2. Neem de titel van de slide over.
3. Neem alle tekstuele inhoud (bullet points, paragrafen) van de slide over.
4. Identificeer waar afbeeldingen of tabellen staan. Gebruik strikte criteria:
   - Informatief: Gebruik de placeholder ALLEEN als de afbeelding tekst, data, grafieken, diagrammen, flowcharts, of een specifiek wetenschappelijk/technisch diagram bevat dat cruciaal is voor begrip van de slide. Formaat: [Informatieve Afb eelding/Tabel: Geef een neutrale beschrijving van wat zichtbaar is of het onderwerp]
   - Decoratief: Gebruik de placeholder voor ALLE foto's van mensen, landschappen, bedrijfslogo's, universiteitslogo's, achtergrondillustraties, stockfoto's, of sfeerbeelden. Bij twijfel, classificeer het als decoratief! Formaat: [Decoratieve Afbeelding]
5. Laat de zin "Share Your talent move the world" weg, indien aanwezig.
6. Lever de output als platte tekst, zonder specifieke Word-opmaak anders dan de slide-indicatie en de placeholders."""

PROMPT_AUDIO_TRANSCRIPTION = """Maak een nauwkeurig en 'schoon' transcript van het bijgevoegde audiobestand.
Instructies:
1. Transcribeer de gesproken tekst zo letterlijk mogelijk.
2. Verwijder stopwoorden en aarzelingen (zoals "eh," "uhm," "nou ja," "weet je wel") om de leesbaarheid te verhogen, maar behoud de volledige inhoudelijke boodschap. Verander geen zinsconstructies.
3. Gebruik geen tijdcodes.
4. Gebruik dlinea's om langere spreekbeurten op te delen.
5. Schrijf de uiteindelijke output volledig in deze taal: {output_language}."""

PROMPT_INTERVIEW_TRANSCRIPTION = """Transcribe this interview in the format: tim

```

ecode (mm:ss) - speaker - caption.

Rules:

- Use speaker A, speaker B, etc. to identify speakers.
- Keep timestamps in each line.
- Write the output fully in this language: {output\_language}."""

PROMPT\_INTERVIEW\_SUMMARY = """You are an expert interviewer analyst.

Create a concise summary of this interview.

Rules:

- Maximum one page equivalent (about 400-600 words).
- Focus only on the most important points, commitments, and conclusions.
- Use short headings and bullet points where useful.
- Do not invent information outside the transcript.
- Write the output fully in this language: {output\_language}.

Transcript:

{transcript}

"""

PROMPT\_INTERVIEW\_SECTIONED = """You are an expert transcript editor.

Rewrite this interview transcript into a structured version with clear headings.

Rules:

- Keep timestamps and speaker labels from the source where possible.
- Split content into relevant sections (for example: Introduction, Background, Key Discussion, Decisions, Next Steps).
- Use meaningful heading titles based on actual content.
- Do not invent information outside the transcript.
- Write the output fully in this language: {output\_language}.

Transcript:

{transcript}

"""

PROMPT\_MERGE\_TEMPLATE = """Creëer een volledige, integrale en goed leesbare uitwerking van een college door de slide-tekst en het audio-transcript naadloos te combineren. Het eindresultaat moet een compleet naslagwerk zijn.

Kernprincipe:

Jouw taak is niet om samen te vatten, maar om te completeren. Het doel is volledigheid, niet beknoptheid. Combineer alle relevante informatie van de slides en de audio tot één compleet, doorlopend en goed gestructureerd document. Wees niet terughoudend met de lengte; de output moet zo lang zijn als nodig is om alle inhoud te dekken. Beschouw het als het uitschrijven van een college voor iemand die er niet bij kon zijn en geen detail mag missen.

Instructies voor Verwerking:

1. Integreer in plaats van te synthetiseren:

- Gebruik de slide-tekst als de ruggengraat en de structuur van het document.
- Verweef de gesproken tekst uit het audio-transcript op de juiste logische plek in de slide-tekst.
- Voeg alle aanvullende uitleg, context, voorbeelden, nuanceringen en zijspraken uit de audio toe. Als de spreker een concept van de slide verder uitlegt, moet die volledige uitleg in de tekst komen.
- Behoud details: Verwijder geen informatie omdat het een 'detail' lijkt. Alle inhoudelijke informatie uit de audio is relevant.

2. Redigeer voor Leesbaarheid (niet voor beknoptheid):

- Verwijder alleen letterlijke herhalingen waarbij de audio exact hetzelfde zegt als de slide-tekst. Als de audio het anders verwoordt, behoud dan de audio-versie omdat deze vaak natuurlijker is.
- Zorg ervoor dat alle overbodige conversationele zinnen (bv. "Oké, dan gaan we nu naar de volgende slide," "Hebben jullie hier vragen over?") en directies aan studenten ("Noteer dit goed," "Dit komt op het tentamen") worden verwijderd, tenzij ze cruciaal zijn voor de context.
- Herschrijf zinnen waar nodig om een vloeierende overgang te creëren tussen de slide-informatie en de toegevoegde audio-uitleg. De tekst moet lezen als één coherente geheel.

### 3. Structuur en Opmaak:

- Gebruik de slide-titels als koppen. Creeëer waar nodig subkoppen voor subonderwerpen die in de audio worden besproken.
- Gebruik alinea's en bullet points om de tekst overzichtelijk en leesbaar te maken.
- Gebruik absoluut geen labels zoals "Audio:", "Spreker:" of "Slide:".
- Zorg voor een professionele, informatieve en neutrale toon.
- Schrijf de uiteindelijke output volledig in deze taal: {output\_language}.

### 4. Omgaan met Visuele Elementen:

- Neem de placeholders voor [Informatieve Afbeelding/Tabel: ...] op de juiste plek in de tekst op.
- Laat placeholders voor [Decoratieve Afbeelding] volledig weg uit de uiteindelijke output.

Input:

1. Slide-tekst:

{slide\_text}

2. Audio-transcript:

{transcript}"""

PROMPT\_STUDY\_TEMPLATE = """You are an expert university professor creating study materials. I will provide you with the complete text of a lecture or slide deck

.

Your task is to generate {flashcard\_amount} flashcards and {question\_amount} multiple-choice test questions based strictly on the provided text. Do not invent outside information.

Write all generated output fully in this language: {output\_language}.

RULES FOR FLASHCARDS:

- The 'front' should be a clear term or concept.
- The 'back' should be a concise, accurate definition/explanation.

RULES FOR TEST QUESTIONS:

- Create challenging, university-level multiple-choice questions.
- Provide exactly 4 options (A, B, C, D) as an array of strings.
- Provide the correct answer (must match one option exactly).
- Provide a brief 'explanation' of WHY the answer is correct.

REQUIRED OUTPUT FORMAT:

You must respond with strictly valid JSON matching this structure:

```
{}  
  "flashcards": [{"front": "string", "back": "string"}],  
  "test_questions": [{"question": "string", "options": ["string", "string", "string", "string"], "answer": "string", "explanation": "string"}]
```

LECTURE TEXT:

{source\_text}

"""

```
# =====  
# FIRESTORE USER FUNCTIONS  
# =====
```

```
def get_or_create_user(uid, email):  
    """Get a user from Firestore, or create them with free credits if they don't  
    exist."""  
    user_ref = db.collection('users').document(uid)  
    user_doc = user_ref.get()  
  
    if user_doc.exists:  
        user_data = user_doc.to_dict()
```

```

# Update email if it changed
if user_data.get('email') != email and email:
    user_ref.update({'email': email})
    user_data['email'] = email
return user_data
else:
    # New user - create with free credits
    user_data = {
        'uid': uid,
        'email': email,
        'lecture_credits_standard': FREE_LECTURE_CREDITS,
        'lecture_credits_extended': 0,
        'slides_credits': FREE_SLIDES_CREDITS,
        'interview_credits_short': FREE_INTERVIEW_CREDITS,
        'interview_credits_medium': 0,
        'interview_credits_long': 0,
        'total_processed': 0,
        'created_at': time.time(),
    }
    user_ref.set(user_data)
    print(f"New user created: {uid} ({email})")
    return user_data

def grant_credits_to_user(uid, bundle_id):
    """Grant credits from a purchased bundle to a user in Firestore."""
    bundle = CREDIT_BUNDLES.get(bundle_id)
    if not bundle:
        print(f"Warning: Unknown bundle_id '{bundle_id}' in grant_credits_to_use
r")
        return False

    user_ref = db.collection('users').document(uid)
    user_doc = user_ref.get()

    if not user_doc.exists:
        # User not in Firestore yet - create with defaults first
        user_data = {
            'uid': uid,
            'email': '',
            'lecture_credits_standard': FREE_LECTURE_CREDITS,
            'lecture_credits_extended': 0,
            'slides_credits': FREE_SLIDES_CREDITS,
            'interview_credits_short': FREE_INTERVIEW_CREDITS,
            'interview_credits_medium': 0,
            'interview_credits_long': 0,
            'total_processed': 0,
            'created_at': time.time(),
        }
        user_ref.set(user_data)

    # Add the purchased credits
    for credit_key, credit_amount in bundle['credits'].items():
        user_ref.update({credit_key: firestore.Increment(credit_amount)})
        print(f"Granted {credit_amount} '{credit_key}' credits to user {uid}.")
```

return True

```

def deduct_credit(uid, credit_type_primary, credit_type_fallback=None):
    """Deduct one credit from the user. Returns the credit type that was deducted, or None if no credits."""
    user_ref = db.collection('users').document(uid)
    user_doc = user_ref.get()
```

```

if not user_doc.exists:
    return None

user_data = user_doc.to_dict()

if user_data.get(credit_type_primary, 0) > 0:
    user_ref.update({
        'credit_type_primary': firestore.Increment(-1),
        'total_processed': firestore.Increment(1),
    })
    return credit_type_primary
elif credit_type_fallback and user_data.get(credit_type_fallback, 0) > 0:
    user_ref.update({
        'credit_type_fallback': firestore.Increment(-1),
        'total_processed': firestore.Increment(1),
    })
    return credit_type_fallback
else:
    return None

def deduct_interview_credit(uid):
    """Deduct one interview credit, checking short -> medium -> long. Returns the credit type deducted, or None."""
    user_ref = db.collection('users').document(uid)
    user_doc = user_ref.get()

    if not user_doc.exists:
        return None

    user_data = user_doc.to_dict()

    if user_data.get('interview_credits_short', 0) > 0:
        user_ref.update({
            'interview_credits_short': firestore.Increment(-1),
            'total_processed': firestore.Increment(1),
        })
        return 'interview_credits_short'
    elif user_data.get('interview_credits_medium', 0) > 0:
        user_ref.update({
            'interview_credits_medium': firestore.Increment(-1),
            'total_processed': firestore.Increment(1),
        })
        return 'interview_credits_medium'
    elif user_data.get('interview_credits_long', 0) > 0:
        user_ref.update({
            'interview_credits_long': firestore.Increment(-1),
            'total_processed': firestore.Increment(1),
        })
        return 'interview_credits_long'
    else:
        return None

def refund_credit(uid, credit_type):
    """Refund one credit back to the user after a failed processing job."""
    if not uid or not credit_type:
        return
    try:
        user_ref = db.collection('users').document(uid)
        user_ref.update({
            credit_type: firestore.Increment(1),
            'total_processed': firestore.Increment(-1),
        })
    
```

```
        })
    print(f"✅ Refunded 1 '{credit_type}' credit to user {uid} due to process
ing failure.")
except Exception as e:
    print(f"❌ Failed to refund credit '{credit_type}' to user {uid}: {e}")

def save_purchase_record(uid, bundle_id, stripe_session_id):
    """Save a purchase record to Firestore for purchase history."""
    bundle = CREDIT_BUNDLES.get(bundle_id)
    if not bundle:
        return
    try:
        db.collection('purchases').add({
            'uid': uid,
            'bundle_id': bundle_id,
            'bundle_name': bundle['name'],
            'price_cents': bundle['price_cents'],
            'currency': bundle['currency'],
            'credits': bundle['credits'],
            'stripe_session_id': stripe_session_id,
            'created_at': time.time(),
        })
        print(f"
```