

## PROJECT TREE

```
interview-agent/
├── .gitignore
├── Dockerfile
├── README.MD
├── app
│   ├── asr.py
│   ├── config.py
│   ├── main.py
│   ├── openai_agent.py
│   ├── prompts.json
│   └── transcriber.py
├── career_coach_beta.py
├── check-devices
│   ├── audio-windows.py
│   └── monitor-check.py
├── db
│   └── db_start.py
├── docker-compose.yml
├── launch.bat
├── project_tree.py
├── recordings
├── repo_doc_to_pdf.py
├── requirements.txt
├── static
│   ├── favicon.ico
│   ├── js.js
│   └── styles.css
├── templates
│   └── index.html
└── windows
    ├── diarization.py
    ├── listener.py
    ├── mic_listener.py
    ├── overlay.py
    ├── overlay_listener.py
    ├── screen.py
    ├── screen_headless.py
    ├── speaker_diarization.py
    ├── stream_listener.py
    └── stream_listener_test.py
```

.gitignore

```
/input_audio/  
/.env  
/recordings/  
/.venv/  
/screenshot.jpg  
/agent_history.db  
/docs/
```

## Dockerfile

```
FROM python:3.10-slim

WORKDIR /app

RUN apt update && apt install -y ffmpeg && pip install --no-cache-dir \
    fastapi uvicorn openai jinja2 python-multipart soundfile requests

COPY ./app /app
COPY ./static /app/static
COPY ./templates /app/templates

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000", "--reload"]
```

## README.MD

```
pip install -r requirements.txt
pip freeze > requirements.txt
streamlit run .\career_coach_beta.py
pip install .\sentencepiece-0.2.1-cp313-cp313-win_amd64.whl
# У файлі .env заміни `your-openai-key-here` на свій ключ
uvicorn app.main:app --reload
# Що можна було зайти в локальній мережі
uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload
# Windows Слухає екран
python windows\listener.py
# Екраний помічник аудіо
python windows/overlay.py
F9

Teams не спрацював
Треба перевірити налаштування Teams
В Slack запрацювало коли я колонки прибрав і поставив CABLE Output

Meets +
Telegram +
Slack +

У розробці (або на майбутнє):
Drag'n'Drop для файлів
Автоматичне виявлення мовця?
Локальний LLM через Ollama
KaiAgentOS інтеграція
Камера або remote screen capturing
```

## career\_coach\_beta.py

```
from dotenv import load_dotenv
import os
from langchain_openai import ChatOpenAI
import streamlit as st
from langchain.prompts import PromptTemplate

load_dotenv() # Loads variables from .env into environment
openai_api_key = os.getenv("OPENAI_API_KEY")
print(f"OpenAI API Key: {openai_api_key}")

llm = ChatOpenAI(model="gpt-4o", api_key=openai_api_key)

prompt_template = PromptTemplate(
    input_variables=["position", "company", "strengths", "weaknesses"],
    template= """You are a career coach. Provide tailored interview tips for the
position of {position} at {company}.
Highlight your strengths in {strengths} and prepare for questions
about your weaknesses such as {weaknesses}."""
)

st.title("Interview Helper")

position = st.text_input("What is your position: ")
company = st.text_input("What is your company: ")
strengths = st.text_area("What are your strengths: ")
weaknesses = st.text_area("What are your weaknesses: ")

if position and company and strengths and weaknesses:
    response = llm.invoke(prompt_template.format(position=position, company=company,
strengths=strengths, weaknesses=weaknesses))
    st.write(response.content)
```

## docker-compose.yml

```
version: "3.8"

services:
  interview-agent:
    container_name: interview-agent
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8000:8000"
    volumes:
      - ./app
    env_file:
      - .env

# interview-agent-listener:
#   container_name: interview-agent-listener
#   build:
#     context: ./listener
#     dockerfile: Dockerfile
#   volumes:
#     - ./input_audio:/app/input_audio
#   env_file:
#     - .env
#   depends_on:
#     - interview-agent
```

launch.bat

```
@echo off
uvicorn app.main:app --reload
python listener.py
pause
```

## project\_tree.py

```
# project_tree.py
import os

EXCLUDE_DIRS = {".idea", ".venv", "__pycache__", ".git"}

def print_tree(startpath, prefix=""):
    try:
        items = sorted(
            name for name in os.listdir(startpath) if name not in EXCLUDE_DIRS
        )
    except FileNotFoundError:
        print(f"Path not found: {startpath}")
        return

    for i, name in enumerate(items):
        path = os.path.join(startpath, name)
        connector = "└─ " if i == len(items) - 1 else "├─ "
        print(prefix + connector + name)
        if os.path.isdir(path):
            extension = "    " if i == len(items) - 1 else "|    "
            print_tree(path, prefix + extension)

if __name__ == "__main__":
    root = "." # шукаємо з поточної директорії
    print(os.path.basename(os.path.abspath(root)) + "/" )
    print_tree(root)
```



## repo\_doc\_to\_pdf.py

```
"""
Usage:
python repo_doc_to_pdf.py --root . --out codebook.pdf --max-bytes 800000 --wrap 100
--root — корінь проекту
--out — шлях до PDF
--max-bytes — великі файли пропускаються
--wrap — ширина переносу рядків у символах
"""
import os, sys, argparse, textwrap, pathlib, unicodedata

import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages

try:
    import chardet
except Exception:
    chardet = None

# Безпечно відрубуюмо TeX/MathText
mpl.rcParams["text.usetex"] = False
mpl.rcParams["mathtext.default"] = "rm"

EXCLUDE_DIRS = {
    ".git", ".venv", "__pycache__", ".idea", ".mypy_cache", ".pytest_cache",
    "node_modules", "dist", "build", ".DS_Store", "input_audio", "outsource", "docs"
}
EXCLUDE_EXTS = {
    ".png", ".jpg", ".jpeg", ".webp", ".gif", ".ico", ".pdf",
    ".zip", ".tar", ".gz", ".7z", ".mp4", ".mov", ".mp3", ".wav"
}

EXCLUDE_FILES = {".env", "celerybeat-schedule", "interview.log", "history.json", "agent_history.db"}

def is_text_file(path: str) -> bool:
    ext = pathlib.Path(path).suffix.lower()
    if ext in EXCLUDE_EXTS:
        return False
    try:
        with open(path, "rb") as f:
            chunk = f.read(4096)
            if not chunk:
                return True
            # справжній нуль-байт
            if b"\x00" in chunk:
                return False
            return True
    except Exception:
        return False

def detect_encoding(data: bytes) -> str:
    if chardet:
        try:
            enc = chardet.detect(data).get("encoding")
            if enc:
                return enc
        except Exception:
            pass
    return "utf-8"

def read_text(path: str, max_bytes: int) -> str:
    with open(path, "rb") as f:
        data = f.read()
    if len(data) > max_bytes:
        return f"[SKIPPED: file too large ({len(data)} bytes)]"
    enc = detect_encoding(data)
    try:
        return data.decode(enc, errors="replace")
    except Exception:
        try:
            return data.decode("utf-8", errors="replace")
        except Exception:
            return data.decode("latin-1", errors="replace")
```

## repo\_doc\_to\_pdf.py

```
def walk_files(root: str):
    for dirpath, dirnames, filenames in os.walk(root):
        dirnames[:] = [d for d in dirnames if d not in EXCLUDE_DIRS and not d.startswith(".tox")]
        for fn in sorted(filenames):
            if fn in EXCLUDE_FILES:
                continue
            full = os.path.join(dirpath, fn)
            rel = os.path.relpath(full, root)
            if not is_text_file(full):
                continue
            yield rel, full

def build_tree_text(root: str) -> str:
    lines = []
    base = os.path.basename(os.path.abspath(root)) or root
    lines.append(f"{base}/")
    def _print_tree(startpath, prefix=""):
        try:
            items = sorted(
                [n for n in os.listdir(startpath)
                 if n not in EXCLUDE_DIRS and n not in EXCLUDE_FILES]
            )
        except FileNotFoundError:
            return
        for i, name in enumerate(items):
            path = os.path.join(startpath, name)
            connector = "└─ " if i == len(items) - 1 else "├─ "
            lines.append(prefix + connector + name)
            if os.path.isdir(path):
                extension = "  " if i == len(items) - 1 else "|  "
                _print_tree(path, prefix + extension)
    _print_tree(root)
    return "\n".join(lines)

def sanitize_text(s: str) -> str:
    """Нормалізація для Matplotlib PDF:
    - заміна всіх '□' → '□' (U+FF04), щоб повністю вимкнути mathtext
    - дроп NULL/контрольні (крім \t \n \r), variation selectors, ZWJ/ZWNJ
    - дроп не-BMP (ємодзі), щоб бекенд PDF не падав
    - мапінг деяких проблемних гліфів на ASCII
    """
    if not s:
        return s
    # повністю вимкнути mathtext
    s = s.replace('□', '□')

    out = []
    for ch in s:
        code = ord(ch)
        if ch == '\x00':
            continue
        cat = unicodedata.category(ch)
        if cat.startswith('C') and ch not in ('\t', '\n', '\r'):
            continue
        if code in (0xFE0F, 0x200D, 0x200C): # VS16, ZWJ, ZWNJ
            continue
        if code > 0xFFFF: # emoji / non-BMP
            continue
        if ch in {'v', 'v'}:
            ch = 'v'
        out.append(ch)
    return ''.join(out)

def add_text_pages(pdf: PdfPages, title: str, text: str, wrap_width=100, font_size=9,
header_size=11):
    # sanitize early
    title = sanitize_text(title)
    text = sanitize_text(text)

    # A4 portrait
    page_w, page_h = 8.27, 11.69
    left_margin, right_margin, top_margin, bottom_margin = 0.5, 0.5, 0.7, 0.7
```

## repo\_doc\_to\_pdf.py

```
usable_height = page_h - top_margin - bottom_margin

# wrap text
wrapped_lines = []
for line in text.splitlines():
    line = sanitize_text(line.expandtabs(4))
    wrapped_lines.extend(textwrap.wrap(line, width=wrap_width, replace_whitespace=False) or
[""])

# simple line-height calc
line_height_in = (font_size * 1.2) / 72.0
lines_per_page = max(1, int(usable_height / line_height_in) - 4)

# render
for page_idx in range(0, len(wrapped_lines) or 1, lines_per_page):
    fig = plt.figure(figsize=(page_w, page_h))
    ax = fig.add_axes([0, 0, 1, 1])
    ax.axis("off")
    # header
    ax.text(0.5, 1 - top_margin / page_h + 0.02, title,
           ha="center", va="top", fontsize=header_size, family="monospace")
    # body
    chunk = wrapped_lines[page_idx: page_idx + lines_per_page]
    body_text = "\n".join(chunk) if chunk else ""
    ax.text(left_margin / page_w, 1 - (top_margin + 0.3) / page_h,
           body_text, ha="left", va="top", fontsize=font_size, family="monospace")
    pdf.savefig(fig, bbox_inches="tight")
    plt.close(fig)

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--root", default=".", help="Project root directory")
    ap.add_argument("--out", default="codebook.pdf", help="Output PDF path")
    ap.add_argument("--max-bytes", type=int, default=800000, help="Skip files larger than this many
bytes")
    ap.add_argument("--wrap", type=int, default=100, help="Characters per line for wrapping")
    args = ap.parse_args()

    tree_text = build_tree_text(args.root)
    with PdfPages(args.out) as pdf:
        add_text_pages(pdf, "PROJECT TREE", tree_text, wrap_width=args.wrap, font_size=9,
header_size=12)
        for rel, full in walk_files(args.root):
            if not is_text_file(full):
                continue
            try:
                content = read_text(full, args.max_bytes)
            except Exception as e:
                content = f"[ERROR reading file: {e}]"
            add_text_pages(pdf, rel, content, wrap_width=args.wrap, font_size=8, header_size=10)

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

## app\asr.py

```
import os
import uuid

# Створимо папку recordings, якщо її ще нема
os.makedirs("recordings", exist_ok=True)

def transcribe_audio(client, audio_bytes: bytes) -> str:
    temp_filename = os.path.join("recordings", f"{uuid.uuid4().hex}.wav")

    try:
        with open(temp_filename, "wb") as f:
            f.write(audio_bytes)

        with open(temp_filename, "rb") as audio_file:
            transcript = client.audio.transcriptions.create(
                model="whisper-1",
                file=audio_file,
            )
        return transcript.text
    finally:
        if os.path.exists(temp_filename):
            os.remove(temp_filename)
```

## app\config.py

```
from openai import OpenAI
import os
from dotenv import load_dotenv

load_dotenv()

def get_openai_client()-> OpenAI:
    return OpenAI(
        api_key=os.getenv("OPENAI_API_KEY")
    )
```

## app\main.py

```
import sys
from dbm import sqlite3

from fastapi import FastAPI, UploadFile, File, Form, Body
from fastapi.responses import HTMLResponse, JSONResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
from fastapi.middleware.cors import CORSMiddleware
from starlette.requests import Request
from app.asr import transcribe_audio
from app.openai_agent import get_answer, session_messages, get_answer_with_image,
capture_screenshot_b64
from app.config import get_openai_client
import json
from pathlib import Path
from pydantic import BaseModel
import subprocess
from db.db_start import init_db, log_to_db, DB_PATH

init_db()

LOG_PATH = Path("history.json")
LOG_PATH.touch(exist_ok=True)

client = get_openai_client()

MODE_PROMPT = "short"

def append_log(question, answer, source="mic"):
    log = []
    if LOG_PATH.exists():
        try:
            log = json.loads(LOG_PATH.read_text().strip() or "[]")
        except Exception:
            log = []
    log.append({
        "question": question,
        "answer": answer,
        "source": source
    })
    LOG_PATH.write_text(json.dumps(log[-100:], indent=2)) # зберігаємо останні 100

app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

templates = Jinja2Templates(directory="templates")
app.mount("/static", StaticFiles(directory="static"), name="static")

@app.get("/", response_class=HTMLResponse)
async def read_root(request: Request):
    return templates.TemplateResponse("index.html", {"request": request})

@app.post("/ask/file")
async def ask_file(file: UploadFile = File(...)):
    audio_bytes = await file.read()
    question = transcribe_audio(client, audio_bytes)
    answer = get_answer(client, question)
    append_log(question, answer, source="mic")
    log_to_db(question, answer, source="mic", model="gpt-4o", mode=MODE_PROMPT)
    return JSONResponse({"question": question, "answer": answer})

@app.post("/ask/audio")
async def ask_audio(file: UploadFile = File(...)):
    audio_bytes = await file.read()
    question = transcribe_audio(client, audio_bytes)
    answer = get_answer(client, question)
    append_log(question, answer, source="mic")
```

## app\main.py

```
log_to_db(question, answer, source="mic", model="gpt-4o", mode=MODE_PROMPT)
return JSONResponse({"question": question, "answer": answer})

class TextRequest(BaseModel):
    text: str

@app.post("/ask/text")
async def ask_text(body: TextRequest):
    question = body.text.strip()
    answer = get_answer(client, question)
    append_log(question, answer, source="text")
    log_to_db(question, answer, source="text", model="gpt-4o", mode=MODE_PROMPT)
    return JSONResponse({"question": question, "answer": answer})

@app.post("/reset")
async def reset_context():
    session_messages.clear()
    LOG_PATH.write_text("[]")
    return {"status": "reset"}

@app.get("/history")
async def get_history():
    if LOG_PATH.exists():
        try:
            content = LOG_PATH.read_text().strip()
            if content:
                return JSONResponse(content=json.loads(content))
        except Exception as e:
            print("Error reading history:", e)
    return JSONResponse(content=[])

listener_proc = None
LISTENER_PATH = str(Path("windows/listener.py").resolve())

@app.post("/listener/start")
async def start_listener():
    global listener_proc
    if listener_proc and listener_proc.poll() is None:
        return {"status": "already running"}
    listener_proc = subprocess.Popen([sys.executable, LISTENER_PATH])
    return {"status": "started"}

@app.post("/listener/stop")
async def stop_listener():
    global listener_proc
    if listener_proc and listener_proc.poll() is None:
        listener_proc.terminate()
        listener_proc = None
    return {"status": "stopped"}
    return {"status": "not running"}

class ScreenRequest(BaseModel):
    prompt: str

SCREEN_PATH = str(Path("windows/screen_headless.py").resolve())

@app.post("/screen/run")
async def run_screen_tool():
    subprocess.Popen([sys.executable, SCREEN_PATH])
    return {"status": "started"}

class ScreenImageRequest(BaseModel):
    image_b64: str
    prompt: str

@app.post("/screen/analyze")
async def screen_analyze(req: ScreenImageRequest):
    answer = get_answer_with_image(client, req.prompt, req.image_b64)
    append_log(req.prompt, answer, source="screen")
    log_to_db(req.prompt, answer, source="screen", model="gpt-4o", mode=MODE_PROMPT)
    return {"question": req.prompt, "answer": answer}
```

## app\main.py

```
PROMPTS_PATH = Path("app/prompts.json")

def load_prompts():
    try:
        return json.loads(PROMPTS_PATH.read_text(encoding="utf-8"))
    except Exception as e:
        print("❌ Помилка при завантаженні prompts.json:", e)
        return {}

@app.post("/mode")
async def set_mode(mode: str = Body(...)):
    global MODE_PROMPT
    MODE_PROMPT = mode
    global current_mode_prompt
    prompts = load_prompts()
    if mode in prompts:
        current_mode_prompt = prompts[mode]
        return {"status": "ok", "mode": mode}
    else:
        return JSONResponse(status_code=400, content={"error": "Невідомий режим"})

MIC_LISTENER_PATH = str(Path("windows/mic_listener.py").resolve())
mic_listener_proc = None

@app.post("/mic/start")
async def start_mic_listener():
    global mic_listener_proc
    if mic_listener_proc and mic_listener_proc.poll() is None:
        return {"status": "already running"}
    mic_listener_proc = subprocess.Popen([sys.executable, MIC_LISTENER_PATH])
    return {"status": "started"}

@app.post("/mic/stop")
async def stop_mic_listener():
    global mic_listener_proc
    if mic_listener_proc and mic_listener_proc.poll() is None:
        mic_listener_proc.terminate()
        mic_listener_proc = None
        return {"status": "stopped"}
    return {"status": "not running"}

OVERLAY_PATH = str(Path("windows/overlay.py").resolve())
@app.post("/overlay", response_class=HTMLResponse)
async def start_overlay():
    subprocess.Popen([sys.executable, OVERLAY_PATH])
    return {"status": "started"}

OVERLAY_LISTENER_PATH = str(Path("windows/overlay_listener.py").resolve())

@app.post("/overlay/listener")
async def start_overlay_listener():
    subprocess.Popen([sys.executable, OVERLAY_LISTENER_PATH])
    return {"status": "started"}

@app.get("/latest")
async def get_latest():
    if LOG_PATH.exists():
        try:
            log = json.loads(LOG_PATH.read_text().strip() or "[]")
            if log:
                return {"question": log[-1]["question"], "answer": log[-1]["answer"]}
        except Exception:
            pass
    return {"question": "", "answer": "❌ Немає відповіді"}

@app.get("/analytics/questions_per_day")
def questions_per_day():
    with sqlite3.connect(DB_PATH) as conn:
        rows = conn.execute('''
            SELECT DATE(timestamp) as day, COUNT(*)
            FROM history
            GROUP BY day
            ORDER BY day DESC
        ''')
```



app\main.py

```
    '').fetchall()  
return [{"day": row[0], "count": row[1]} for row in rows]
```

## app\openai\_agent.py

```
from datetime import datetime
import base64
import cv2
import numpy as np
from PIL import ImageGrab

session_messages = []

current_mode_prompt = "Дай коротку відповідь і приклад з коду (якщо доречно)."
```

```
def get_system_prompt():
    return {
        "role": "system",
        "content": (
            f"Ти AI-асистент. {current_mode_prompt} "
            "Відповідай на питання користувача."
        )
    }

def get_answer(client, question: str) -> str:
    user_msg = {"role": "user", "content": question}

    # Додати в історію
    session_messages.append(user_msg)

    # Починаємо prompt з system + останніх N повідомлень (наприклад, 4)
    context = [get_system_prompt()] + session_messages[-4:]

    response = client.chat.completions.create(
        model="gpt-4o",
        messages=context,
        temperature=0.9
    )

    assistant_msg = {
        "role": "assistant",
        "content": response.choices[0].message.content.strip()
    }

    # Додаємо відповідь у контекст
    session_messages.append(assistant_msg)

    # логування у файл
    with open("interview.log", "a", encoding="utf-8") as f:
        f.write(f"{datetime.now()}\nQ: {question}\nA: {assistant_msg['content']}\n\n")

    return assistant_msg["content"]

def get_answer_with_image(client, prompt: str, image_b64: str) -> str:
    messages = [
        {
            "role": "system",
            "content": (
                "Ти аналітик. Користувач надіслав питання і зображення. "
                "Використай зображення при відповіді. Відповідай коротко і технічно."
            )
        },
        {
            "role": "user",
            "content": [
                {"type": "text", "text": prompt},
                {"type": "image_url", "image_url": {"url": f"data:image/jpeg;base64,{image_b64}"}}
            ]
        }
    ]

    response = client.chat.completions.create(
        model="gpt-4o",
        messages=messages,
        temperature=0.9
    )
```

## app\openai\_agent.py

```
assistant_msg = {
    "role": "assistant",
    "content": response.choices[0].message.content.strip()
}

session_messages.append(assistant_msg)
return assistant_msg["content"]

def capture_screenshot_b64() -> str:
    screenshot = ImageGrab.grab()
    screenshot_np = cv2.cvtColor(np.array(screenshot), cv2.COLOR_RGB2BGR)
    _, buffer = cv2.imencode(".jpg", screenshot_np)
    return base64.b64encode(buffer).decode()
```

## app\prompts.json

```
{
  "short": "Ти – досвідчений інженер зі співбесіди. Відповідай коротко, по суті, впевнено, без води. Стиль – як у старшого розробника, який точно знає, що говорить. Уникай зайвих пояснень, якщо вони не критичні.",
  "code": "Ти – технічний ментор. Дай коротку, чітку відповідь, а якщо доречно – одразу додай приклад коду. Не пояснюй зайвого: достатньо короткого коду та короткої пояснюючої фрази (якщо потрібно). Форматуй код як у StackOverflow.",
  "hr": "Уяви, що ти пояснюєш це не технічній людині – наприклад, рекрутеру або HR-менеджеру. Використовуй прості слова, аналогії, зрозумілі навіть без технічної підготовки. Уникай професійного жаргону. Твоя мета – бути зрозумілим.",
  "long": "Ти – викладач або сеньйор-розробник, який хоче глибоко пояснити суть. Відповідай розгорнуто: дай загальну ідею, приклади, код (якщо доречно), контекст, можливі варіанти реалізації. Мета – щоб навіть джун зрозумів.",
  "screen": "Ти – технічний інтерв'юер. Проаналізуй зображення з екрану. Якщо бачиш на ньому технічне або тестове питання – коротко відповідай на нього, вибери правильний варіант (якщо є), і поясни чому. Якщо це фрагмент коду – поясни його суть або виправ помилки. Відповідай стисло й по суті."
}
```

## app\transcriber.py

```
import openai
import requests
import os

openai.api_key = os.environ["OPENAI_API_KEY"]

def transcribe_and_send(filepath):
    with open(filepath, "rb") as f:
        transcript = openai.Audio.transcribe("whisper-1", f)
    q = transcript["text"]
    print("", q)
    requests.post("http://interview-agent:8000/ask", files={"file": open(filepath, "rb")})
```

## check-devices\audio-windows.py

```
# Подивитись список аудіопристроїв (Windows)
import sounddevice as sd

print("\n=== Аудіопристрої ===")
for i, device in enumerate(sd.query_devices()):
    print(f"[{i}] {device['name']} ({device['hostapi']})")

print("\n=== Перевірити, які sample rates підтримує device_id=31 ===")
sd.check_input_settings(device=31, samplerate=48000) # або 44100, 48000

print("\n=== [PaErrorCode -9997] тоді перебирай samplerate ===")
```

## check-devices\monitor-check.py

```
# import screeninfo
#
# for m in screeninfo.get_monitors():
#     print(f"Monitor {m.name}: {m.width}x{m.height} at ({m.x}, {m.y})")

import mss
import numpy as np
from PIL import Image

with mss.mss() as sct:
    monitor = sct.monitors[1] # або [2] – другий екран
    sct_img = sct.grab(monitor)
    img = Image.frombytes("RGB", sct_img.size, sct_img.rgb)
    img.save("screenshot.jpg")
```

## db\db\_start.py

```
import sqlite3
from datetime import datetime

DB_PATH = "agent_history.db"

def init_db():
    with sqlite3.connect(DB_PATH) as conn:
        conn.execute('''
            CREATE TABLE IF NOT EXISTS history (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                timestamp TEXT,
                source TEXT,
                question TEXT,
                answer TEXT,
                model TEXT,
                mode TEXT
            );
        ''')

def log_to_db(question, answer, source="text", model="gpt-4o", mode="short"):
    with sqlite3.connect(DB_PATH) as conn:
        conn.execute('''
            INSERT INTO history (timestamp, source, question, answer, model, mode)
            VALUES (?, ?, ?, ?, ?, ?)
        ''', (
            datetime.now().isoformat(),
            source,
            question,
            answer,
            model,
            mode
        ))
```



## static\js.js

```
document.addEventListener("DOMContentLoaded", () => {
  const form = document.getElementById("upload-form");
  const fileInput = document.getElementById("audio");
  const questionEl = document.getElementById("question-text");
  const answerEl = document.getElementById("answer-text");
  const historyContainer = document.getElementById("history-list");
  const resetBtn = document.getElementById("reset-btn");
  const recordBtn = document.getElementById("record-btn");

  let mediaRecorder;
  let audioChunks = [];
  let currentHistory = [];

  function renderAnswerMarkdown(rawText) {
    const answerHtml = marked.parse(rawText || "");
    answerEl.innerHTML = `<div class="rendered-answer">{answerHtml}</div>`;
    if (typeof hljs !== "undefined") {
      answerEl.querySelectorAll("pre code").forEach((el) => {
        hljs.highlightElement(el);
      });
    }
  }

  // Завантажити історію при старті
  fetch("/history")
    .then((res) => res.json())
    .then((history) => {
      currentHistory = history;
      updateHistory(currentHistory);
    });

  resetBtn.addEventListener("click", () => {
    fetch("/reset", {method: "POST"})
      .then((res) => res.json())
      .then(() => {
        questionEl.innerText = "";
        answerEl.innerText = "";
        currentHistory = [];
        updateHistory([]);
      });
  });

  form.addEventListener("submit", async (e) => {
    e.preventDefault();

    if (!fileInput.files.length) {
      questionEl.innerText = "";
      answerEl.innerText = "⚠ Оберіть аудіофайл!";
      return;
    }

    const formData = new FormData();
    formData.append("file", fileInput.files[0]);

    questionEl.innerText = "🔄 Обробка...";
    answerEl.innerText = "";

    const response = await fetch("/ask/file", {
      method: "POST",
      body: formData,
    });

    const data = await response.json();
    questionEl.innerText = data.question;
    renderAnswerMarkdown(data.answer);

    currentHistory.push({question: data.question, answer: data.answer});
    updateHistory(currentHistory);
  });

  // Кнопка запису
  recordBtn.addEventListener("mousedown", startRecording);
});
```

## static\js.js

```
recordBtn.addEventListener("mouseup", stopRecording);

function startRecording() {
  if (!navigator.mediaDevices || !navigator.mediaDevices.getUserMedia) {
    answerEl.innerHTML = "⚠ Браузер не підтримує мікрофон або доступ заборонено.";
    return;
  }

  navigator.mediaDevices.getUserMedia({audio: true}).then((stream) => {
    audioChunks = [];
    mediaRecorder = new MediaRecorder(stream);
    mediaRecorder.ondataavailable = (e) => audioChunks.push(e.data);
    mediaRecorder.onstop = sendRecording;
    mediaRecorder.start();
    recordBtn.innerHTML = "Запис...";
  }).catch((err) => {
    answerEl.innerHTML = "❌ Доступ до мікрофона заборонено.";
    console.error("getUserMedia error:", err);
  });
}

function stopRecording() {
  mediaRecorder.stop();
  recordBtn.innerHTML = "Запис";
}

function sendRecording() {
  const audioBlob = new Blob(audioChunks, {type: "audio/webm"});
  const formData = new FormData();
  formData.append("file", audioBlob, "recording.webm");

  if (audioBlob.size < 1128) {
    answerEl.innerHTML = "⚠ Аудіо занадто коротке або порожнє!";
    return;
  }

  questionEl.innerHTML = "🔄 Обробка...";
  answerEl.innerHTML = "";

  fetch("/ask/audio", {
    method: "POST",
    body: formData,
  })
    .then((res) => res.json())
    .then((data) => {
      questionEl.innerHTML = data.question;
      renderAnswerMarkdown(data.answer);
      if (ttsEnabled) speakText(data.answer);
      currentHistory.push({question: data.question, answer: data.answer});
      updateHistory(currentHistory);
    });
}

function updateHistory(history) {
  historyContainer.innerHTML = "";
  [...history].reverse().forEach((entry) => {
    const block = document.createElement("div");
    const answerHtml = marked.parse(entry.answer || "");
    block.classList.add("history-entry");
    block.innerHTML = `
<div><strong>Q:</strong> ${entry.question}</div>
<div><strong>A:</strong><div class="rendered-answer">${answerHtml}</div></div>
`;
    if (typeof hljs !== "undefined") {
      block.querySelectorAll("pre code").forEach((el) => {
        hljs.highlightElement(el);
      });
    }
    historyContainer.appendChild(block);
  });
}
```

## static\js.js

```
const toggleBtn = document.getElementById("theme-toggle");

const theme = localStorage.getItem("theme");
if (theme === "dark") {
    document.documentElement.classList.add("dark");
    toggleBtn.textContent = "";
}

toggleBtn.addEventListener("click", () => {
    const isDark = document.documentElement.classList.toggle("dark");
    localStorage.setItem("theme", isDark ? "dark" : "light");
    toggleBtn.textContent = isDark ? "" : "";
});

function speakTextVoices() {
    window.speechSynthesis.getVoices().forEach(v => console.log(v.lang, v.name));
}

let ttsEnabled = false;
let selectedLang = "en-US";

const langSelect = document.getElementById("voice-lang-select");
langSelect.addEventListener("change", () => {
    selectedLang = langSelect.value;
    localStorage.setItem("ttsLang", selectedLang); // зберігаємо в локалсторедж
});

if (localStorage.getItem("ttsLang")) {
    selectedLang = localStorage.getItem("ttsLang");
    langSelect.value = selectedLang;
}

const repeatBtn = document.getElementById("repeat-tts-btn");
let lastAnswerSpoken = "";

const ttsBtn = document.getElementById("tts-toggle-btn");

ttsBtn.addEventListener("click", () => {
    ttsEnabled = !ttsEnabled;
    ttsBtn.textContent = ttsEnabled ? " Не озвучувати" : " Озвучити";
    repeatBtn.style.display = ttsEnabled ? "inline-block" : "none";

    // speakTextVoices();
});

repeatBtn.addEventListener("click", () => {
    if (lastAnswerSpoken) speakText(lastAnswerSpoken);
});

async function getVoice(preferredLang) {
    return new Promise((resolve) => {
        const fallbackLang = "en-US";
        const pickVoice = () => {
            const voices = speechSynthesis.getVoices();
            let voice = voices.find(v => v.lang === preferredLang);
            if (!voice) {
                console.warn(`⚠ Голос ${preferredLang} не знайдено. Використовую
${fallbackLang}`);
                alert(`⚠ Голос ${preferredLang} не знайдено. Використовую ${fallbackLang}`);
                voice = voices.find(v => v.lang === fallbackLang) || voices[0];
            }
            resolve(voice);
        };
        if (speechSynthesis.getVoices().length) pickVoice();
        else speechSynthesis.onvoiceschanged = pickVoice;
    });
}

async function speakText(text) {
    const voice = await getVoice(selectedLang);
    const utterance = new SpeechSynthesisUtterance(text);
    utterance.voice = voice;
}
```

## static\js.js

```
utterance.rate = 1.0;
utterance.pitch = 1.0;
window.speechSynthesis.speak(utterance);
lastAnswerSpoken = text;
}

const sendTextBtn = document.getElementById("send-text-btn");
const manualInput = document.getElementById("manual-question");

sendTextBtn.addEventListener("click", async () => {
  const text = manualInput.value.trim();
  if (!text) return;

  questionEl.innerText = text;
  answerEl.innerText = "⏱ Обработка...";

  const response = await fetch("/ask/text", {
    method: "POST",
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify({text})
  });

  const data = await response.json();
  renderAnswerMarkdown(data.answer);
  if (ttsEnabled) await speakText(data.answer);

  currentHistory.push({question: text, answer: data.answer});
  updateHistory(currentHistory);

  manualInput.value = "";
});

const listenerBtn = document.getElementById("listener-toggle-btn");
let listenerProcess = null;
let isListenerRunning = false;
let listenerHistoryInterval = null;

function loadHistory() {
  fetch("/history")
    .then(res => res.json())
    .then((history) => {
      currentHistory = history;
      updateHistory(currentHistory);
    });
}

listenerBtn.addEventListener("click", async () => {
  if (!isListenerRunning) {
    // Запуск listener через бекенд
    const res = await fetch("/listener/start", {method: "POST"});
    const data = await res.json();
    if (data.status === "started") {
      isListenerRunning = true;
      listenerBtn.textContent = "Зупинити Listener";
      listenerHistoryInterval = setInterval(loadHistory, 3000);
    } else {
      alert("⚠ Не вдалося запустити listener");
    }
  } else {
    // Зупинка listener
    const res = await fetch("/listener/stop", {method: "POST"});
    const data = await res.json();
    if (data.status === "stopped") {
      isListenerRunning = false;
      listenerBtn.textContent = "Запустити Listener";
      // Зупиняємо інтервал
      if (listenerHistoryInterval) {
        clearInterval(listenerHistoryInterval);
        listenerHistoryInterval = null;
      }
    }
  }
});
```

## static\js.js

```
    }

    // Завантажити остаточну історію
    loadHistory();
  } else {
    alert("⚠ Не вдалося зупинити listener");
  }
}
});

const screenBtn = document.getElementById("screen-analyze-btn");

screenBtn.addEventListener("click", async () => {
  const res = await fetch("/screen/run", {
    method: "POST"
  });

  if (res.ok) {
    // почекаємо кілька секунд, поки screen.py зробить свій запит
    setTimeout(loadHistory, 30000);
  } else {
    alert("❌ Не вдалося запустити screen.py");
  }
});

const modeSelect = document.getElementById("mode-select");
modeSelect.addEventListener("change", async () => {
  const mode = modeSelect.value;

  const res = await fetch("/mode", {
    method: "POST",
    headers: {"Content-Type": "application/json"},
    body: JSON.stringify(mode)
  });

  if (res.ok) {
    console.log("🔄 Режим змінено:", mode);
  } else {
    alert("❌ Не вдалося змінити режим");
  }
});

const micBtn = document.getElementById("mic-listener-toggle-btn");
let isMicRunning = false;
let micInterval = null;

micBtn.addEventListener("click", async () => {
  if (!isMicRunning) {
    const res = await fetch("/mic/start", {method: "POST"});
    const data = await res.json();
    if (data.status === "started") {
      isMicRunning = true;
      micBtn.textContent = "Зупинити мікрофон";
      micInterval = setInterval(loadHistory, 3000);
    } else {
      alert("❌ Не вдалося запустити mic_listener");
    }
  } else {
    const res = await fetch("/mic/stop", {method: "POST"});
    const data = await res.json();
    if (data.status === "stopped") {
      isMicRunning = false;
      micBtn.textContent = "Live з мікрофона";
      clearInterval(micInterval);
    } else {
      alert("❌ Не вдалося зупинити mic_listener");
    }
  }
});

document.getElementById("overlay-btn").addEventListener("click", async () => {
  const res = await fetch("/overlay", {method: "POST"});
  if (res.ok) {

```

## static\js.js

```
        console.log(" Overlay запущено");
    } else {
        // alert("❌ Не вдалося запустити overlay");
        console.log("❌ Не вдалося запустити overlay");
    }
});

document.getElementById("overlay-listener-btn").addEventListener("click", async () => {
    const res = await fetch("/overlay/listener", {method: "POST"});
    if (res.ok) {
        console.log(" Overlay Listener запущено");
    } else {
        alert("❌ Не вдалося запустити overlay");
    }
});

setInterval(() => {
    fetch("/history")
        .then(res => res.json())
        .then((history) => {
            // Якщо нова історія відрізняється – оновити
            if (JSON.stringify(history) !== JSON.stringify(currentHistory)) {
                currentHistory = history;
                updateHistory(currentHistory);
            }
        });
}, 3000); // кожні 3 секунди
});
```

## static\styles.css

```
:root {
  --bg: #f4f6f8;
  --text: #222;
  --card: #ffffff;
  --accent: #007bff;
  --accent-hover: #0056b3;
  --border: #ddd;
  --entry-bg: #f1f3f5;
  --entry-border: #007bff;
}

:root.dark {
  --bg: #121212;
  --text: #eaeaea;
  --card: #1e1e1e;
  --accent: #4dabf7;
  --accent-hover: #339af0;
  --border: #444;
  --entry-bg: #1c1c1c;
  --entry-border: #4dabf7;
}

body {
  font-family: "Segoe UI", sans-serif;
  margin: 0;
  padding: 0;
  background: var(--bg);
  color: var(--text);
  display: flex;
  justify-content: center;
  align-items: start;
  min-height: 100vh;
}

.container {
  max-width: 90%;
  width: 100%;
  margin: 2rem auto;
  background: var(--card);
  border-radius: 12px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.08);
  padding: 2rem;
}

h1 {
  text-align: center;
  margin-bottom: 1.5rem;
  color: var(--text);
}

/* .controls {*/
/*   display: flex;*/
/*   gap: 1rem;*/
/*   flex-wrap: wrap;*/
/*   margin-bottom: 1.5rem;*/
/*   justify-content: center;*/
/* }*/

button,
input[type="file"] {
  padding: 0.6rem 1rem;
  font-size: 1rem;
  border: none;
  border-radius: 6px;
  cursor: pointer;
}

button {
  background-color: var(--accent);
  color: white;
  transition: background 0.2s ease;
}
```

## static\styles.css

```
button:hover {
    background-color: var(--accent-hover);
}

input[type="file"] {
    background-color: #eee;
    color: #333;
}

.qa-display {
    background: var(--entry-bg);
    border-radius: 8px;
    padding: 1rem;
    margin-bottom: 1.5rem;
    border: 1px solid var(--border);
}

.qa-display p {
    margin: 0.5rem 0;
    word-wrap: break-word;
}

.qa-display strong {
    color: var(--accent);
}

.history-section {
    margin-top: 1rem;
}

.history-section h2 {
    font-size: 1.2rem;
    margin-bottom: 0.8rem;
    color: var(--text);
}

.history-list {
    height: calc(100vh - 250px); /* 200px – це відступ для шапки, кнопок тощо */
    overflow-y: auto;
    display: flex;
    flex-direction: column;
    gap: 0.75rem;
}

.history-entry {
    background: var(--entry-bg);
    padding: 0.75rem;
    border-left: 4px solid var(--entry-border);
    border-radius: 8px;
    box-shadow: 0 1px 3px rgba(0, 0, 0, 0.05);
}

.history-entry div {
    margin-bottom: 0.25rem;
    line-height: 1.4;
}

.history-entry strong {
    color: var(--accent);
}

@media (max-width: 600px) {
    .controls {
        flex-direction: column;
        align-items: stretch;
    }

    .container {
        padding: 1rem;
    }
}

/*#voice-lang-select {*/
```



## static\styles.css

```
/* position: absolute;*/
/* top: 15px;*/
/* right: 80px;*/
/* padding: 0.4rem 0.6rem;*/
/* border-radius: 6px;*/
/* border: 1px solid var(--border);*/
/* background: var(--card);*/
/* color: var(--text);*/
/* font-size: 0.9rem;*/
/* font-family: "Segoe UI", sans-serif;*/
/* box-shadow: 0 1px 3px rgba(0, 0, 0, 0.05);*/
/* appearance: none;*/
/*}*/

#voice-lang-select:focus {
  outline: none;
  border-color: var(--accent);
}

#voice-lang-select {
  background-image: linear-gradient(45deg, transparent 50%, var(--text) 50%),
    linear-gradient(135deg, var(--text) 50%, transparent 50%);
  background-position: calc(100% - 18px) calc(1em + 2px), calc(100% - 13px) calc(1em + 2px);
  background-size: 5px 5px, 5px 5px;
  background-repeat: no-repeat;
  padding-right: 2rem;
}

#manual-question {
  padding: 0.6rem;
  font-size: 1rem;
  border-radius: 6px;
  border: 1px solid var(--border);
  background: var(--card);
  color: var(--text);
  flex-grow: 1;
  min-width: 200px;
}

/* .theme-toggle {*/
/* position: absolute;*/
/* top: 10px;*/
/* right: 20px;*/
/* cursor: pointer;*/
/* font-size: 1.2rem;*/
/* background: none;*/
/* border: none;*/
/* color: var(--text);*/
/*}*/

.layout-grid {
  display: grid;
  grid-template-columns: 1fr 1.2fr;
  gap: 2rem;
}

.left-panel {
  display: flex;
  flex-direction: column;
  gap: 1rem;
}

.right-panel {
  max-height: 100%;
  overflow-y: auto;
}

@media (max-width: 900px) {
  .layout-grid {
    grid-template-columns: 1fr;
  }
}
```

## static\styles.css

```
.rendered-answer code {
  font-family: "Fira Code", monospace;
  font-size: 0.95rem;
}

.rendered-answer pre {
  background: #1e1e1e;
  padding: 0.75rem;
  border-radius: 8px;
  overflow-x: auto;
}

.header-controls {
  display: flex;
  justify-content: flex-end;
  gap: 1rem;
  margin-bottom: 1rem;
  align-items: center;
}

.header-controls select,
.header-controls button {
  padding: 0.4rem 0.6rem;
  font-size: 0.95rem;
  border-radius: 6px;
  border: 1px solid var(--border);
  background: var(--card);
  color: var(--text);
  font-family: "Segoe UI", sans-serif;
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.05);
  appearance: none;
  cursor: pointer;
}

.header-controls button.theme-toggle {
  border: none;
  background: none;
  font-size: 1.2rem;
  box-shadow: none;
}

.controls {
  display: flex;
  flex-direction: column;
  gap: 1rem;
  margin-bottom: 1.5rem;
}

.controls-group {
  display: flex;
  flex-wrap: wrap;
  gap: 1rem;
  align-items: center;
  justify-content: flex-start;
}

textarea#manual-question {
  width: 100%;
  min-height: 80px;
  resize: vertical;
  padding: 0.6rem;
  font-size: 1rem;
  border-radius: 6px;
  border: 1px solid var(--border);
  background: var(--card);
  color: var(--text);
  font-family: inherit;
}
```

```

<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8"/>
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <link rel="icon" type="image/x-icon" href="/static/favicon.ico">
  <title>AI Interview Assistant</title>
  <script src="/static/outsource/marked.min.js"></script>
  <!-- Підсвітка коду -->
  <link rel="stylesheet"
    href="/static/outsource/github-dark.min.css">
  <script src="/static/outsource/highlight/highlight.min.js"></script>
  <link rel="stylesheet" href="/static/styles.css"/>
</head>
<body>
<div class="container">
  <div class="header-controls">
    <select id="mode-select" title="Режим відповіді">
      <option value="short"> Коротко</option>
      <option value="code"> 3 кодом</option>
      <option value="hr"> Для HR</option>
      <option value="long"> Розгорнуто</option>
    </select>
    <select id="voice-lang-select" title="Оберіть мову озвучення">
      <option value="uk-UA"> Українська</option>
      <option value="ru-RU"> Російська</option>
      <option value="pl-PL"> Польська</option>
      <option value="en-US"> Англійська</option>
    </select>
    <button class="theme-toggle" id="theme-toggle" title="Змінити тему"></button>
  </div>
  <h1> AI Interview Assistant</h1>
  <div class="layout-grid">
    <div class="left-panel">
      <!-- КОНТРОЛІ -->
      <div class="controls">
        <div class="controls-group">
          <!-- Текст + відправка -->
          <textarea id="manual-question" placeholder="Введіть своє питання..."></textarea>
          <div class="row">
            <form id="upload-form">
              <button type="button" id="send-text-btn"> Відправити текст</button>
              <input type="file" id="audio" name="audio" accept="audio/*"
style="display: none;">
              <button type="submit"> Надіслати файл</button>
            </form>
          </div>
        </div>
        <div class="controls-group">
          <!-- Аудіо -->
          <button id="record-btn"> Запис</button>
          <button id="tts-toggle-btn"> Озвучити</button>
          <button id="repeat-tts-btn" style="display: none;"> Повторити голосом</button>
        </div>
        <div class="controls-group">
          <!-- Live режими -->
          <button id="listener-toggle-btn"> Запустити Listener</button>
          <button id="mic-listener-toggle-btn"> Live з мікрофона</button>
        </div>
        <div class="controls-group">
          <!-- Інше -->
          <button id="screen-analyze-btn"> Аналіз екрана</button>
          <button id="reset-btn"> Очистити контекст</button>
        </div>
        <div class="controls-group">
          <!-- Overlay -->
          <button id="overlay-btn"> Показати Overlay</button>
          <button id="overlay-listener-btn"> Показати Overlay Listener</button>
        </div>
      </div>
    </div>
  </div>

```

## templates\index.html

```
<div class="qa-display">
  <p><strong> Q:</strong> <span id="question-text"></span></p>
  <p><strong> A:</strong> <span id="answer-text"></span></p>
</div>
<div class="right-panel">
  <!-- ICTOPIA -->
  <div class="history-section">
    <h2> Ictopia</h2>
    <div id="history-list" class="history-list"></div>
  </div>
</div>
</div>
<script src="/static/js.js"></script>
</body>
</html>
```

## windows\diarization.py

```
import os
import uuid
from pyannote.audio import Pipeline
from tempfile import NamedTemporaryFile
import whisper
from dotenv import load_dotenv

load_dotenv()
hf_token = os.getenv("HUGGINGFACE_TOKEN")

# Завантажуємо модель diarization (один раз)
pipeline = Pipeline.from_pretrained("pyannote/speaker-diarization", use_auth_token=hf_token)

# Завантажуємо Whisper
whisper_model = whisper.load_model("base")

def save_temp_wav(audio_bytes: bytes) -> str:
    temp_filename = f"temp_{uuid.uuid4().hex}.wav"
    with open(temp_filename, "wb") as f:
        f.write(audio_bytes)
    return temp_filename

def extract_questions_from_diarization(audio_path: str, speaker_label: str = None):
    # 1. Отримати сегменти з diarization
    diarization = pipeline(audio_path)

    # 2. Транскрибувати через Whisper
    result = whisper_model.transcribe(audio_path)
    segments = result["segments"] # [{'start': ..., 'end': ..., 'text': ...}]

    # 3. Поєднати дані: кому належить який сегмент
    speaker_segments = []
    for seg in segments:
        for turn, _, speaker in diarization.itertracks(yield_label=True):
            if turn.start <= seg["start"] <= turn.end:
                speaker_segments.append({
                    "speaker": speaker,
                    "text": seg["text"].strip(),
                    "start": seg["start"],
                    "end": seg["end"]
                })
        break

    # 4. Якщо speaker_label заданий – залишаємо лише його
    if speaker_label:
        speaker_segments = [s for s in speaker_segments if s["speaker"] == speaker_label]

    # 5. Витягаємо лише ті сегменти, які є питаннями
    questions = [s["text"] for s in speaker_segments if s["text"].strip().endswith("?")]

    return questions
```

## windows\listener.py

```
import sounddevice as sd
import soundfile as sf
import time
import os
import requests

output_dir = "../input_audio"
os.makedirs(output_dir, exist_ok=True)
API_URL = "http://localhost:8000/ask/audio" # або інший порт, якщо змінив

# Ці функції для постійного прослуховування колонок/екрану

# Отримуємо список хост-API (для перетворення індексу на назву)
hostapis = sd.query_hostapis()
devices = sd.query_devices()

# Знайти WASAPI + CABLE Output
for i, d in enumerate(devices):
    hostapi_index = d["hostapi"]
    hostapi_name = hostapis[hostapi_index]["name"]
    if "CABLE Output" in d["name"] and "WASAPI" in hostapi_name:
        sd.default.device = (i, None)
        print(f" Використовую пристрій: [{i}] {d['name']} ({hostapi_name})")
        break
else:
    print("❌ Не знайдено CABLE Output через WASAPI")
    exit(1)

def record_audio(duration=2, samplerate=48000):
    print(" Recording...")
    audio = sd.rec(int(duration * samplerate), samplerate=samplerate, channels=1, dtype='int16')
    sd.wait()
    timestamp = int(time.time())
    filename = os.path.join(output_dir, f"recording_{timestamp}.wav")
    sf.write(filename, audio, samplerate)
    print(f" Saved: {filename}")
    return filename

def send_audio_to_api(filename):
    with open(filename, "rb") as f:
        files = {"file": (filename, f, "audio/wav")}
        response = requests.post(API_URL, files=files)
        if response.status_code == 200:
            print(f" Відповідь: {response.json()['answer']}")
        else:
            print(f"❌ Помилка API: {response.status_code} {response.text}")

while True:
    fname = record_audio()
    send_audio_to_api(fname)
    time.sleep(0.2)
```

## windows\mic\_listener.py

```
import sounddevice as sd
import soundfile as sf
import time
import os
import requests
import uuid

output_dir = "../input_audio"
os.makedirs(output_dir, exist_ok=True)

API_URL = "http://localhost:8000/ask/audio"

# Ці функції для постійного прослуховування мікрофона

def record_audio(duration=4, samplerate=16000):
    print(" Запис з мікрофона...")
    audio = sd.rec(int(duration * samplerate), samplerate=samplerate, channels=1, dtype='int16')
    sd.wait()
    filename = os.path.join(output_dir, f"mic_{uuid.uuid4().hex}.wav")
    sf.write(filename, audio, samplerate)
    return filename

def send_audio_to_api(filename):
    with open(filename, "rb") as f:
        response = requests.post(API_URL, files={"file": f})
        if response.status_code == 200:
            print(f" Відповідь: {response.json()['answer']}")
        else:
            print(f" Помилка API: {response.status_code} {response.text}")
    os.remove(filename)

while True:
    try:
        fname = record_audio()
        send_audio_to_api(fname)
        time.sleep(1)
    except KeyboardInterrupt:
        print(" Зупинено вручну.")
        break
```

## windows\overlay.py

```
import subprocess
import tkinter as tk
import sounddevice as sd
import soundfile as sf
import threading
import requests
import os
import uuid
import keyboard
import pyttsx3
import time
import numpy as np

API_URL = "http://localhost:8000/ask/audio"
SAMPLERATE = 16000
CHANNELS = 1

# □ ОБЕРИ свою мову і голос:
LANG_CODE = "ru" # або 'en', 'pl', 'ru' і т.д.
VOICE_NAME = "Irina"

# Цей клас для запису голосом питань в вигляді програми поверх всіх вікон

class OverlayAssistant:
    def __init__(self):
        self.root = tk.Tk()
        self.root.overridereirect(True)
        self.root.attributes("-topmost", True)
        self.root.attributes("-alpha", 0.9)
        self.root.configure(bg="black")

        self.offset_x = 0
        self.offset_y = 0

        self.engine = pyttsx3.init()
        self.voice = self.pick_voice(LANG_CODE, VOICE_NAME)

        self.text_label = tk.Label(self.root, text=" Тримай F9 щоб говорити | Ctrl+F9 – озвучити",
font=("Segoe UI", 11),
                                bg="black", fg="lime", wraplength=800, justify="left")
        self.text_label.pack(padx=10, pady=(10, 5))

        self.button_frame = tk.Frame(self.root, bg="black")
        self.button_frame.pack(pady=(0, 10))

        self.screen_button = tk.Button(self.button_frame, text=" Аналіз екрана",
command=self.run_screen_headless,
                                font=("Segoe UI", 9), bg="gray20", fg="white")
        self.screen_button.pack(side="left", padx=5)

        self.tts_button = tk.Button(self.button_frame, text=" Озвучити", command=self.speak_text,
                                font=("Segoe UI", 9), bg="gray20", fg="white")
        self.tts_button.pack(side="left", padx=5)

        self.close_button = tk.Button(self.button_frame, text="✖", command=self.root.destroy,
                                font=("Segoe UI", 9), bg="darkred", fg="white")
        self.close_button.pack(side="left", padx=5)

        self.root.bind("<ButtonPress-1>", self.start_move)
        self.root.bind("<B1-Motion>", self.do_move)

        self.last_answer = ""
        self.recording = False
        self.audio_frames = []

        threading.Thread(target=self.listen_hotkey_loop, daemon=True).start()

        self.root.geometry("+60+60")

        self.hidden = False

        self.screen_recording = False
```



## windows\overlay.py

```
self.screen_audio_frames = []

def pick_voice(self, lang_code, voice_name=""):
    voices = self.engine.getProperty("voices")
    matched = []
    for v in voices:
        langs = v.languages[0] if isinstance(v.languages[0], str) else
v.languages[0].decode(errors="ignore")
        if lang_code.lower() in langs.lower() or lang_code.lower() in v.id.lower():
            matched.append(v)
    if voice_name:
        matched = [v for v in matched if voice_name.lower() in v.name.lower()]
    return matched[0] if matched else self.engine.getProperty("voice")

def speak_text(self):
    if not self.last_answer:
        return
    self.engine.setProperty("voice", self.voice.id)
    self.engine.setProperty("rate", 175)
    self.engine.say(self.last_answer)
    self.engine.runAndWait()

def listen_hotkey_loop(self):
    self.text_label.config(text=" Тримай F9 щоб говорити | Ctrl+F9 – озвучити")
    keyboard.add_hotkey("F8", self.run_screen_headless)
    keyboard.add_hotkey("ctrl+F9", self.speak_text)
    keyboard.add_hotkey("F11", self.toggle_visibility)
    while True:
        if keyboard.is_pressed("F9"):
            self.start_recording()
            while keyboard.is_pressed("F9"):
                time.sleep(0.1)
            self.stop_and_send()

        # elif keyboard.is_pressed("F10"):
        #     self.start_screen_recording()
        #     while keyboard.is_pressed("F10"):
        #         time.sleep(0.1)
        #     self.stop_and_send_screen()

        elif keyboard.is_pressed("F10"):
            self.start_ffmpeg_recording()
            while keyboard.is_pressed("F10"):
                time.sleep(0.1)
            self.stop_and_send_ffmpeg()

        time.sleep(0.05)

def start_ffmpeg_recording(self):
    self.text_label.config(text=" FFMPEG запис активний... Відпустіть F7 щоб відправити")
    self.root.lift()
    self.root.update()
    self.ffmpeg_filename = f"temp_{uuid.uuid4().hex}.wav"
    self.ffmpeg_proc = subprocess.Popen([
        "ffmpeg",
        "-f", "dshow",
        "-i", "audio=CABLE Output (VB-Audio Virtual Cable)",
        "-acodec", "pcm_s16le",
        "-ar", "48000",
        "-ac", "1",
        self.ffmpeg_filename
    ], stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)

def stop_and_send_ffmpeg(self):
    if not hasattr(self, "ffmpeg_proc"):
        return
    self.text_label.config(text=" Завершення ffmpeg запису...")
    try:
        self.ffmpeg_proc.communicate(timeout=2)
    except subprocess.TimeoutExpired:
        self.ffmpeg_proc.terminate()
    time.sleep(0.5)
```

## windows\overlay.py

```
if not os.path.exists(self.ffmpeg_filename):
    self.text_label.config(text="⚠ Файл не записано")
    return

self.text_label.config(text="Відправка (ffmpeg)...")
try:
    with open(self.ffmpeg_filename, "rb") as f:
        response = requests.post(API_URL, files={"file": f})
        if response.status_code == 200:
            self.last_answer = response.json().get("answer", "Нема відповіді")
            self.text_label.config(text=f"✅ {self.last_answer}")
        else:
            self.text_label.config(text=f"❌ Статус: {response.status_code}")
except Exception as e:
    self.text_label.config(text=f"⚠ Помилка: {e}")
finally:
    if os.path.exists(self.ffmpeg_filename):
        os.remove(self.ffmpeg_filename)

def run_screen_headless(self):
    self.text_label.config(text="Аналіз екрана...")
    self.root.lift()
    self.root.update()

    try:
        output = subprocess.check_output(
            ["python", "screen_headless.py"],
            stderr=subprocess.STDOUT,
            encoding="utf-8" # <- важливо
        ).strip()

        if "❌ Відповідь:" in output:
            answer = output.split("❌ Відповідь:")[1].strip()
            self.last_answer = answer
            self.text_label.config(text=f"✅ {answer}")
        elif output:
            self.text_label.config(text=output)
        else:
            self.text_label.config(text="⚠ Порожня відповідь від screen_headless")

    except subprocess.CalledProcessError as e:
        self.text_label.config(text=f"❌ Помилка запуску: {e.output.strip()}")
    except Exception as e:
        self.text_label.config(text=f"⚠ Виняток: {e}")

def toggle_visibility(self):
    if self.hidden:
        self.root.deiconify()
        self.hidden = False
    else:
        self.root.withdraw()
        self.hidden = True

def start_screen_recording(self):
    if self.screen_recording:
        return
    self.text_label.config(text="Запис звуку з екрана...")
    self.root.lift()
    self.root.update()
    self.screen_audio_frames = []
    self.screen_recording = True
    threading.Thread(target=self._screen_record_loop, daemon=True).start()

def _screen_record_loop(self):
    try:
        with sd.InputStream(samplerate=48000, channels=1, dtype='int16') as stream:
            while self.screen_recording:
                data, _ = stream.read(1024)
                self.screen_audio_frames.append(data.copy())
    except Exception as e:
        self.text_label.config(text=f"⚠ Екран запис – помилка: {e}")
        self.screen_recording = False
```

## windows\overlay.py

```
def stop_and_send_screen(self):
    if not self.screen_recording:
        return
    self.screen_recording = False
    self.text_label.config(text="⏏ Обробка звуку з екрана...")
    self.root.update()

    if not self.screen_audio_frames:
        self.text_label.config(text="⚠ Нічого не записано з екрана")
        return

    audio_np = np.concatenate(self.screen_audio_frames, axis=0)
    filename = f"screen_{uuid.uuid4().hex}.wav"
    sf.write(filename, audio_np, samplerate=48000)

    try:
        with open(filename, "rb") as f:
            response = requests.post(API_URL, files={"file": f})
            if response.status_code == 200:
                self.last_answer = response.json().get("answer", "Нема відповіді")
                self.text_label.config(text=f"📄 {self.last_answer}")
            else:
                self.text_label.config(text=f"📄 Статус: {response.status_code}")
    except Exception as e:
        self.text_label.config(text=f"⚠ Помилка: {e}")
    finally:
        if os.path.exists(filename):
            os.remove(filename)

def start_recording(self):
    if self.recording:
        return
    self.text_label.config(text="Запис активний... Відпустіть F9 щоб відправити")
    self.root.lift() # Повертаємо overlay поверх, навіть якщо він загубився
    self.root.update()
    self.audio_frames = []
    self.recording = True
    threading.Thread(target=self._record_loop, daemon=True).start()

def _record_loop(self):
    with sd.InputStream(samplerate=SAMPLERATE, channels=CHANNELS, dtype='int16',
callback=self.audio_callback):
        while self.recording:
            time.sleep(0.1)

def audio_callback(self, indata, frames, time_info, status):
    self.audio_frames.append(indata.copy())

def stop_and_send(self):
    if not self.recording:
        return
    self.recording = False
    self.text_label.config(text="⏏ Обробка...")
    self.root.update()

    if not self.audio_frames:
        self.text_label.config(text="⚠ Нічого не записано")
        return

    audio_np = np.concatenate(self.audio_frames, axis=0)
    filename = f"temp_{uuid.uuid4().hex}.wav"
    sf.write(filename, audio_np, samplerate=SAMPLERATE)

    try:
        with open(filename, "rb") as f:
            response = requests.post(API_URL, files={"file": f})
            if response.status_code == 200:
                self.last_answer = response.json().get("answer", "Нема відповіді")
                self.text_label.config(text=f"📄 {self.last_answer}")
            else:
                self.text_label.config(text=f"📄 Статус: {response.status_code}")
    except Exception as e:
        self.text_label.config(text=f"⚠ Помилка: {e}")
```

## windows\overlay.py

```
finally:
    if os.path.exists(filename):
        os.remove(filename)

# self.root.after(10000, lambda: self.text_label.config(text=" Тримай F9 щоб говорити |
Ctrl+F9 – озвучити"))

def start_move(self, event):
    self.offset_x = event.x
    self.offset_y = event.y

def do_move(self, event):
    x = event.x_root - self.offset_x
    y = event.y_root - self.offset_y
    self.root.geometry(f'+{x}+{y}')

def run(self):
    self.root.mainloop()

if __name__ == "__main__":
    print(" Список голосів у системі:")
    eng = pyttsx3.init()
    for v in eng.getProperty("voices"):
        langs = v.languages[0].decode(errors="ignore") if isinstance(v.languages[0], bytes) else
v.languages[0]
        print(f"    {v.name} | {v.id} | lang: {langs}")
    print("\n Обери потрібну мову і вкажи у LANG_CODE / VOICE_NAME у файлі overlay.py\n")

    OverlayAssistant().run()
```

## windows\overlay\_listener.py

```
import tkinter as tk
import requests
import threading
import time
```

# Цей клас для зображення останньої відповіді у вигляді програми поверх усіх вікон

```
class OverlayListener:
    def __init__(self):
        self.root = tk.Tk()
        self.root.overridereDIRECT(True)
        self.root.attributes("-topmost", True)
        self.root.attributes("-alpha", 0.9)
        self.root.configure(bg="black")

        self.offset_x = 0
        self.offset_y = 0
        self.answer = ""

        self.label = tk.Label(
            self.root,
            text=" Чекаю...",
            font=("Segoe UI", 11),
            bg="black",
            fg="lime",
            wraplength=300,
            justify="left"
        )
        self.label.pack(padx=10, pady=(10, 5))

        self.close_button = tk.Button(
            self.root, text="✖", command=self.root.destroy,
            font=("Segoe UI", 9), bg="darkred", fg="white"
        )
        self.close_button.pack(pady=(0, 10))

        # Drag support
        self.root.bind("<ButtonPress-1>", self.start_move)
        self.root.bind("<B1-Motion>", self.do_move)

        self.root.geometry("+100+100")

        # Start background updater
        threading.Thread(target=self.poll_latest, daemon=True).start()

    def start_move(self, event):
        self.offset_x = event.x
        self.offset_y = event.y

    def do_move(self, event):
        x = event.x_root - self.offset_x
        y = event.y_root - self.offset_y
        self.root.geometry(f'+{x}+{y}')

    def poll_latest(self):
        while True:
            try:
                res = requests.get("http://localhost:8000/latest", timeout=2)
                if res.ok:
                    data = res.json()
                    if data["answer"] != self.answer:
                        self.answer = data["answer"]
                        self.label.config(text=f" {self.answer}")
            except Exception as e:
                self.label.config(text=f"⚠ {e}")
            time.sleep(3)

    def run(self):
        self.root.mainloop()

if __name__ == "__main__":
    OverlayListener().run()
```

## windows\screen.py

```
import tkinter as tk
import requests
import base64
import io
import mss
from PIL import Image

API_URL = "http://localhost:8000/screen/analyze"

system_prompt = "Ти – технічний співбесідник. Проаналізуй цю частину екрана. Якщо бачиш на екрані питання (тестове або програмне), коротко дай відповідь і поясни чому. Якщо є варіанти – вибери правильний."

# Цей клас для знімок з екрана з перетягуванням між екранами
class ScreenTool:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title(" Screen Assistant")
        self.root.geometry("400x80")
        self.root.attributes("-topmost", True)
        self.root.resizable(False, False)

        tk.Label(self.root, text="Перемістіть це вікно на потрібний монітор").pack(pady=5)
        tk.Button(self.root, text=" Спитати", command=self.capture_and_send).pack(pady=5)

        self.root.protocol("WM_DELETE_WINDOW", self.root.destroy)

    def capture_and_send(self):
        self.root.withdraw()
        self.root.after(300, self._do_capture)

    def _do_capture(self):
        with mss.mss() as sct:
            monitor = sct.monitors[1] # [1] = перший повний екран, [2] – другий і т.д.

            sct_img = sct.grab(monitor)
            img = Image.frombytes("RGB", sct_img.size, sct_img.rgb)
            img.save("screenshot.jpg")

            buffered = io.BytesIO()
            img.save(buffered, format="JPEG")
            img_b64 = base64.b64encode(buffered.getvalue()).decode()

        try:
            res = requests.post(API_URL, json={
                "image_b64": img_b64,
                "prompt": system_prompt
            })

            if res.ok:
                print("□ Відповідь:", res.json()["answer"])
            else:
                print("□ API Error:", res.status_code, res.text)
        except Exception as e:
            print("△ Виняток:", e)

        self.root.destroy()

    def run(self):
        self.root.mainloop()

if __name__ == "__main__":
    ScreenTool().run()
```

## windows\screen\_headless.py

```
import requests
import base64
import io
import mss
from PIL import Image

API_URL = "http://localhost:8000/screen/analyze"

system_prompt = "Ти – технічний співбесідник. Проаналізуй цю частину екрана. Якщо бачиш на екрані питання (тестове або програмне), коротко дай відповідь і поясни чому. Якщо є варіанти – вибери правильний."

# Цей клас для знімок з екрана
def capture_and_send(prompt=system_prompt):
    try:
        with mss.mss() as sct:
            monitor = sct.monitors[1]
            sct_img = sct.grab(monitor)
            img = Image.frombytes("RGB", sct_img.size, sct_img.rgb)

            buffered = io.BytesIO()
            img.save(buffered, format="JPEG")
            img_b64 = base64.b64encode(buffered.getvalue()).decode()

            res = requests.post(API_URL, json={
                "image_b64": img_b64,
                "prompt": prompt
            })

            if res.ok:
                print("Відповідь:", res.json()["answer"])
            else:
                print("API Error:", res.status_code, res.text)

    except Exception as e:
        print("Виняток:", e)

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

## windows\speaker\_diarization.py

```
import os
from dotenv import load_dotenv
from pyannote.audio import Pipeline

load_dotenv()
hf_token = os.getenv("HUGGINGFACE_TOKEN")
print(hf_token)
pipeline = Pipeline.from_pretrained(
    "pyannote/speaker-diarization",
    use_auth_token=hf_token
)

def diarize(file_path: str):
    diarization = pipeline(file_path)
    segments = []

    for turn, _, speaker in diarization.itertracks(yield_label=True):
        segments.append({
            "speaker": speaker,
            "start": turn.start,
            "end": turn.end
        })

    return segments

# Пример використання
if __name__ == "__main__":
    audio_file = "your_audio.wav"
    segs = diarize(audio_file)
    for s in segs:
        print(f"[{s['start']:.2f}--{s['end']:.2f}] {s['speaker']}")
```



## windows\stream\_listener.py

```
import sounddevice as sd
import numpy as np
import threading
import requests
import io
import time
import keyboard
import soundfile as sf
import uuid

# === НАЛАШТУВАННЯ ===
API_URL = "http://localhost:8000/ask/audio"
SAMPLE_RATE = 48000
CHANNELS = 1

recording = False
recorded_frames = []

print(" Ініціалізація Push-to-Hold Listener")

# Знайти індекс CABLE Output (WASAPI)
hostapis = sd.query_hostapis()
devices = sd.query_devices()
device_index = None
for i, d in enumerate(devices):
    hostapi_index = d["hostapi"]
    hostapi_name = hostapis[hostapi_index]["name"]
    if "CABLE Output" in d["name"] and "WASAPI" in hostapi_name:
        device_index = i
        print(f" Використовую пристрій: [{i}] {d['name']} ({hostapi_name})")
        break
else:
    raise RuntimeError(" CABLE Output через WASAPI не знайдено")

def send_audio(frames):
    audio_np = np.concatenate(frames, axis=0)
    buffer = io.BytesIO()
    sf.write(buffer, audio_np, samplerate=SAMPLE_RATE, format='WAV')
    buffer.seek(0)

    try:
        response = requests.post(API_URL, files={"file": (f"kai_{uuid.uuid4().hex}.wav", buffer,
"audio/wav")})
        if response.ok:
            print(" Відповідь:", response.json().get("answer"))
        else:
            print(" API помилка:", response.status_code, response.text)
    except Exception as e:
        print("⚠ Виняток при надсиланні:", e)

def record_loop():
    global recording, recorded_frames
    def callback(indata, frames, time_info, status):
        if status:
            print("⚠ Статус потоку:", status)
        if recording:
            recorded_frames.append(indata.copy())

    with sd.InputStream(device=device_index, samplerate=SAMPLE_RATE, channels=CHANNELS,
dtype='int16', callback=callback):
        while True:
            time.sleep(0.05)

def keyboard_loop():
    global recording, recorded_frames
    print("== Утримуй F8 щоб записати, відпусти щоб надіслати")
    while True:
        keyboard.wait("F8")
        print(" Запис... (утримуй)")
        recorded_frames = []
        recording = True
        while keyboard.is_pressed("F8"):
            time.sleep(0.1)
```

windows\stream\_listener.py

```
recording = False
if recorded_frames:
    print(f" Відправка {len(recorded_frames)} фрагментів")
    send_audio(recorded_frames)
else:
    print("⚠ Нічого не записано")

threading.Thread(target=record_loop, daemon=True).start()
keyboard_loop()
```

windows\stream\_listener\_test.py

```
import subprocess
import threading
import time
import keyboard
import requests
import uuid
import os

API_URL = "http://localhost:8000/ask/audio"
OUTPUT_FILE = f"temp_{uuid.uuid4().hex}.wav"

def record_audio_ffmpeg():
    return subprocess.Popen([
        "ffmpeg",
        "-f", "dshow",
        "-i", "audio=CABLE Output (VB-Audio Virtual Cable)",
        "-t", "00:00:30",
        "-acodec", "pcm_s16le",
        "-ar", "48000",
        "-ac", "1",
        OUTPUT_FILE
    ], stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)

def send_audio(filepath):
    try:
        with open(filepath, "rb") as f:
            response = requests.post(API_URL, files={"file": (os.path.basename(filepath), f,
"audio/wav")})
            if response.ok:
                print(" Відповідь:", response.json().get("answer"))
            else:
                print(" API помилка:", response.status_code, response.text)
    except Exception as e:
        print(" Виняток:", e)
    finally:
        if os.path.exists(filepath):
            os.remove(filepath)

print(" FFMPEG Listener активний")
print("== Утримуй F8 щоб записати звук з CABLE Output...")

while True:
    keyboard.wait("F8")
    print(" Запис з CABLE Output...")
    start_time = time.time() # старт запису

    proc = record_audio_ffmpeg()
    while keyboard.is_pressed("F8"):
        time.sleep(0.1)

    print(" Завершення запису...")
    try:
        proc.communicate(timeout=2)
    except subprocess.TimeoutExpired:
        proc.terminate()

    time.sleep(0.3) # щоб файл дописався

    if os.path.exists(OUTPUT_FILE):
        print(" Відправка...")
        send_start = time.time()
        send_audio(OUTPUT_FILE)
        end_time = time.time()

        print(f" Запис: {send_start - start_time:.2f} сек | Відповідь: {end_time - send_start:.2f}
сек | Загалом: {end_time - start_time:.2f} сек")
    else:
        print(" Файл не записано!")
```