GenAI for Accessibility — Backend Scaffold (FastAPI)

A beginner-friendly, copy-pasteable backend scaffold for the hackathon MVP: **PDF → Text extraction → Summarization → TTS → (optional) Translation & Quiz** . This document contains the full project structure, step-by-step setup instructions, and the complete code for each file.

✅What you will get in this scaffold

- A working FastAPI backend with endpoints:
- `/extract` — upload PDF, returns extracted text
- `/summarize` — send text, returns bullet points + ELI5 summary
- `/tts` — convert text summary (or text) to MP3 and return file
- `/translate` — (optional) translate text using a simple translator wrapper
- `/quiz` — generate MCQs & flashcards from a text chunk
- Helper services for OCR, LLM calls, TTS, and chunking
- A clear README with commands to run everything locally

✔️Prerequisites

- Python 3.10+ installed
- `pip` available
- Tesseract OCR installed on your machine:
- **Ubuntu/Debian:** `sudo apt update && sudo apt install -y tesseract-ocr libtesseract-dev`
- **macOS (Homebrew):** `brew install tesseract`
- **Windows:** download installer from https://github.com/tesseract-ocr/tesseract (install and add to PATH)
- (Optional) An OpenAI API key if you want to use the OpenAI LLM. Set `OPENAI_API_KEY` in your environment.

Project structure

```
genai-accessibility/
├─ api/
│  ├─ app/
```

```
│  │  ├─ main.py
│  │  ├─ config.py
│  │  ├─ routes/
│  │  │  ├─ extract.py
│  │  │  ├─ summarize.py
│  │  │  ├─ tts.py
│  │  │  ├─ translate.py
│  │  │  └─ quiz.py
│  │  ├─ services/
│  │  │  ├─ ocr.py
│  │  │  ├─ llm.py
│  │  │  ├─ tts_service.py
│  │  │  ├─ translate_service.py
│  │  │  └─ quiz_service.py
│  │  ├─ models/
│  │  │  └─ schemas.py
│  │  └─ utils/
│  │     └─ chunker.py
│  ├─ requirements.txt
│  └─ Dockerfile   (optional)
└─ README.md
```

File contents (copy-paste these into files exactly as shown)

`api/requirements.txt`

```
fastapi
uvicorn[standard]
python-multipart
pydantic
pypdf
pytesseract
Pillow
pymupdf
openai
gTTS
python-dotenv
requests
deep-translator
```

> `deep-translator` is optional and used as a simple translator; you can remove it if you plan to use another provider.

`api/app/config.py`

```
import os
from dotenv import load_dotenv
load_dotenv()

OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")  # set this in your .env
TEMP_DIR = os.getenv("TEMP_DIR", "/tmp")
TESSERACT_CMD = os.getenv("TESSERACT_CMD")
```

If you installed tesseract but it's not on PATH on Windows, set `TESSERACT_CMD` to the executable path.

`api/app/main.py`

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from routes import extract, summarize, tts, translate, quiz

app = FastAPI(title="GenAI Accessibility API")

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

app.include_router(extract.router, prefix="/extract")
app.include_router(summarize.router, prefix="/summarize")
app.include_router(tts.router, prefix="/tts")
app.include_router(translate.router, prefix="/translate")
app.include_router(quiz.router, prefix="/quiz")

@app.get("/")
async def root():
    return {"message": "GenAI Accessibility API — up and running"}

# To run: uvicorn app.main:app --reload --port 8000
```

---

api/app/utils/chunker.py

```
# Simple safe chunker for long text so we don't send entire book to LLM at once

def chunk_text(text: str, max_chars: int = 3000):
    """Yield chunks roughly max_chars long, cutting at sentence boundaries when
possible."""
    import re
    text = text.strip()
    if len(text) <= max_chars:
        yield text
        return
    # try split on double newlines or sentence ends
    parts = re.split(r"(\n\n|(?<=[.!?])\s+)", text)
    cur = ""
    for p in parts:
        if len(cur) + len(p) > max_chars:
            if cur:
                yield cur.strip()
            cur = p
        else:
            cur += p
    if cur.strip():
        yield cur.strip()
```

---

api/app/services/ocr.py

```
import os
from config import TEMP_DIR, TESSERACT_CMD
from pypdf import PdfReader
import fitz  # PyMuPDF
from PIL import Image
import pytesseract

if TESSERACT_CMD:
    pytesseract.pytesseract.tesseract_cmd = TESSERACT_CMD


def extract_text_from_pdf(path: str) -> str:
    """Try native PDF text extraction first, otherwise fall back to OCR via
PyMuPDF + pytesseract."""
    text_chunks = []
    # 1) try pypdf text extraction
    try:
```

```
        reader = PdfReader(path)
        for page in reader.pages:
            ptxt = page.extract_text() or ""
            text_chunks.append(ptxt)
    except Exception:
        pass

    joined = "\n".join(text_chunks).strip()
    # if extraction yields little text, fallback to OCR
    if len(joined) < 200:
        # load pages as images via fitz
        doc = fitz.open(path)
        ocr_text = []
        for page in doc:
            pix = page.get_pixmap(dpi=200)
            mode = "RGBA" if pix.alpha else "RGB"
            img = Image.frombytes(mode, [pix.width, pix.height], pix.samples)
            page_text = pytesseract.image_to_string(img)
            ocr_text.append(page_text)
        return "\n".join(ocr_text).strip()

    return joined
```

---

api/app/services/llm.py

```
import os
import openai
from config import OPENAI_API_KEY

openai.api_key = OPENAI_API_KEY

DEFAULT_MODEL = os.getenv("LLM_MODEL", "gpt-4o-mini")

SUMMARY_PROMPT = """
You are an educational assistant. Receive a chunk of textbook text and return a
JSON with two fields:
1) "key_points": an array of 5 concise bullet points (each <= 20 words)
2) "eli5": a simple explanation written in plain easy-to-read language (2-4
short paragraphs)
Return output ONLY as valid JSON.

Text:
"""
```

```
def summarize_text(text: str) -> str:
    prompt = SUMMARY_PROMPT + text
    resp = openai.ChatCompletion.create(
        model=DEFAULT_MODEL,
        messages=[{"role": "user", "content": prompt}],
        temperature=0.2,
        max_tokens=600,
    )
    out = resp["choices"][0]["message"]["content"]
    return out


QUIZ_PROMPT = """
Create:
- 4 MCQs (A-D) with correct answer marked
- 3 True/False statements with answers
- 5 flashcards (Q|A)
Return result as JSON with keys: mcqs, tf, flashcards.
Base content ONLY on the following text:
"""

def generate_quiz(text: str) -> str:
    prompt = QUIZ_PROMPT + text
    resp = openai.ChatCompletion.create(
        model=DEFAULT_MODEL,
        messages=[{"role": "user", "content": prompt}],
        temperature=0.3,
        max_tokens=700,
    )
    return resp["choices"][0]["message"]["content"]
```

Note: If you prefer a different LLM provider, replace `openai` calls with their SDK.

`api/app/services/tts_service.py`

```
import os
import uuid
from gtts import gTTS
from config import TEMP_DIR


def text_to_mp3(text: str, lang: str = "en") -> str:
    """Return path to generated MP3 file"""
    filename = f"{uuid.uuid4()}.mp3"
    out_path = os.path.join(TEMP_DIR, filename)
```

```
    tts = gTTS(text=text, lang=lang, slow=False)
    tts.save(out_path)
    return out_path
```

`api/app/services/translate_service.py` **(optional)**

```
from deep_translator import GoogleTranslator


def translate_text(text: str, target_lang: str = "hi") -> str:
    # target_lang: 'hi', 'kn', 'ta', etc.
    translated = GoogleTranslator(target=target_lang).translate(text)
    return translated
```

The `deep-translator` package is a simple wrapper for many translators and is good for prototyping. For production use, prefer an official paid translation API.

`api/app/services/quiz_service.py`

```
from llm import generate_quiz


def make_quiz(text: str) -> dict:
    raw = generate_quiz(text)
    # LLM returns JSON string — try to parse safely
    import json
    try:
        return json.loads(raw)
    except Exception:
        return {"raw": raw}
```

`api/app/models/schemas.py`

```
from pydantic import BaseModel
from typing import Optional

class ExtractResponse(BaseModel):
    text: str

class SummarizeRequest(BaseModel):
```

```
    text: str

class SummarizeResponse(BaseModel):
    summary: str

class TTSRequest(BaseModel):
    text: str
    lang: Optional[str] = "en"

class TranslateRequest(BaseModel):
    text: str
    target: Optional[str] = "hi"

class QuizRequest(BaseModel):
    text: str
```

`api/app/routes/extract.py`

```
from fastapi import APIRouter, UploadFile, File, HTTPException
from starlette.responses import JSONResponse
from services.ocr import extract_text_from_pdf
import tempfile

router = APIRouter()

@router.post("/", response_class=JSONResponse)
async def extract(file: UploadFile = File(...)):
    if not file.filename.endswith(".pdf"):
        raise HTTPException(status_code=400, detail="Please upload a PDF file.")
    # write to a temp file
    tmp = tempfile.NamedTemporaryFile(delete=False, suffix=".pdf")
    content = await file.read()
    tmp.write(content)
    tmp.flush()
    tmp.close()
    text = extract_text_from_pdf(tmp.name)
    return {"text": text}
```

`api/app/routes/summarize.py`

```
from fastapi import APIRouter, Body
from services.llm import summarize_text
```

```
router = APIRouter()

@router.post("/", tags=["summarize"])
async def summarize(body: dict = Body(...)):
    text = body.get("text", "")
    if not text:
        return {"error": "text required"}
    out = summarize_text(text)
    return {"summary": out}
```

`api/app/routes/tts.py`

```
from fastapi import APIRouter, Body
from fastapi.responses import FileResponse
from services.tts_service import text_to_mp3

router = APIRouter()

@router.post("/", tags=["tts"] )
async def tts(body: dict = Body(...)):
    text = body.get("text")
    lang = body.get("lang", "en")
    if not text:
        return {"error": "text required"}
    path = text_to_mp3(text, lang=lang)
    return FileResponse(path, media_type="audio/mpeg", filename="narration.mp3")
```

`api/app/routes/translate.py` **(optional)**

```
from fastapi import APIRouter, Body
from services.translate_service import translate_text

router = APIRouter()

@router.post("/", tags=["translate"])
async def translate(body: dict = Body(...)):
    text = body.get("text")
    target = body.get("target", "hi")
    if not text:
        return {"error": "text required"}
```

```
    translated = translate_text(text, target)
    return {"translated": translated}
```

`api/app/routes/quiz.py`

```
from fastapi import APIRouter, Body
from services.quiz_service import make_quiz

router = APIRouter()

@router.post("/", tags=["quiz"])
async def quiz(body: dict = Body(...)):
    text = body.get("text")
    if not text:
        return {"error": "text required"}
    q = make_quiz(text)
    return {"quiz": q}
```

`api/Dockerfile` **(optional)**

```
FROM python:3.11-slim
WORKDIR /app
COPY api/requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
COPY api/app ./app
ENV PYTHONUNBUFFERED=1
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

✅Step-by-step setup (copy these commands into your terminal)

1. Clone repo or create folder and copy files (use the structure above).

2. Create a virtual environment and activate it:

```
cd genai-accessibility/api
python -m venv .venv
# mac/linux
source .venv/bin/activate
```

```
# windows
.venv\Scripts\activate
```

3. Install dependencies

```
pip install -r requirements.txt
```

4. Create a `.env` file in `api/app/` and add:

```
OPENAI_API_KEY=sk-...   # if using OpenAI
TEMP_DIR=/tmp
# Optional: TESSERACT_CMD=C:\Program Files\Tesseract-OCR\tesseract.exe
```

5. Start the API

```
uvicorn app.main:app --reload --port 8000
```

Open http://localhost:8000/ to verify the service is running.

6. Test endpoints quickly using `curl` (examples):

7. Extract:

```
curl -X POST "http://localhost:8000/extract/" -F "file=@/path/to/
sample.pdf"
```

8. Summarize:

```
curl -X POST "http://localhost:8000/summarize/" -H "Content-Type:
application/json" -d '{"text":"Your long text here"}'
```

9. TTS (download mp3):

```
curl -X POST "http://localhost:8000/tts/" -H "Content-Type: application/
json" -d '{"text":"Hello students", "lang":"en"}' --output narration.mp3
```

10. Quiz:

```
curl -X POST "http://localhost:8000/quiz/" -H "Content-Type: application/
json" -d '{"text":"Text to make quiz from"}'
```

Tips for debugging (common beginner issues)

- **Tesseract not found**: ensure `tesseract` is installed and on PATH or set `TESSERACT_CMD` to the exe location.
- **OpenAI auth error**: make sure `OPENAI_API_KEY` is in `.env` and you restarted the server.
- **Large files**: extract only first few pages for quick testing.
- **API rate limits**: use short chunks for summarization and avoid sending whole books in single request.

Next steps / stretch work (nice to add)

- Add caching of generated summaries/audio (hash input and store outputs).
- Add a small React frontend (upload UI, play audio, toggle dyslexic mode).
- Add authentication & user quotas.
- Replace gTTS with ElevenLabs or Azure TTS for better voices.

If you want help running this locally

I can either: - Paste any single file content here in the chat for you to copy (if you prefer). OR - Walk you through running each step in your terminal and help fix errors you encounter.

Good luck — open the `GenAI-Accessibility-Backend-Scaffold` document in the canvas to view & copy the entire scaffold code. When you're ready, tell me which file you'd like to run first and I will guide you step-by-step.