PRD: “ReadMe” — Local AI Reading Assistant

1. Purpose and Vision

ReadMe is a **self-hosted reading application** that converts digital books and documents (`.pdf`, `.epub`, `.txt`, `.docx`) into natural-sounding speech and intelligent summaries.
It is designed for **personal, local use** — running on your PC — with **optional cloud compute on Heroku** for resource-intensive AI tasks.

The intent is to build a **Speechify-like desktop utility** that prioritizes:

- privacy and ownership of data,
- local responsiveness,
- zero external dependencies except for API calls you explicitly configure.

---

2. Functional Overview

Feature	Description
File Input	Open `.pdf`, `.epub`, `.txt`, `.docx` files from local drive.
Text Extraction	Parse text and structure (pages, chapters, paragraphs).
Speech Synthesis	Convert text to speech using AI voices (via local Coqui-TTS or OpenAI API through Heroku).
Playback Controls	Play, pause, seek, adjust playback speed, repeat paragraph, or skip.
Library Management	Maintain local list of previously opened books with reading progress.
Bookmarks / Notes	Save reading position, annotations, and highlights locally.
AI Summarization	Summarize selected text or chapters (processed on Heroku).

Offline Mode	Fully functional offline reading (except for remote AI services).
Minimal Interface	Lightweight desktop GUI with file explorer, playback, and settings panels.

---

3. System Architecture

Overview

- **Frontend + Local App:** Electron + React (desktop client).
- **Local Backend:** Python FastAPI service running on localhost for parsing, TTS control, and storage.
- **Remote Backend (Heroku):** Handles large AI workloads like text summarization, OpenAI TTS, or Whisper transcription.

Architecture Layers

A. Local Components

Layer	Tech	Purpose
Desktop Shell	Electron + React	GUI for book browsing, playback, and settings.
Local API Server	FastAPI (Python)	Hosts endpoints for local logic and file parsing.
Database	SQLite	Store library metadata, progress, notes, and settings.
File Parsing	PyMuPDF / pdfplumber / ebooklib	Extracts text + metadata from documents.
Local TTS (optional)	Coqui-TTS	Perform on-device text-to-speech when available.
Audio Engine	PyAudio / Web Audio API	Stream audio to local output.

B. Heroku Cloud Components

Service	Tech	Role

Heroku Web App	FastAPI (Python) / Flask	Receives text from local app, returns summarized or TTS-generated audio.
AI Layer	OpenAI API, Whisper, Transformers	Summarization, translation, advanced speech synthesis.
Heroku Postgres (optional)	Used only for caching API results or analytics if desired.	

---

4. Data Flow

1. **User Opens File**
    - Electron app reads file metadata and sends to local FastAPI.
    - FastAPI extracts text and structures it (JSON: `{chapter, page, paragraph}`).
2. **Speech Generation**
    - For small jobs → Coqui-TTS locally.
    - For large jobs → sends text to Heroku FastAPI endpoint (`/generate_audio`).
    - Heroku service calls OpenAI TTS or another model, streams back `.wav`/`.mp3`.
3. **Playback**
    - Local player streams from local disk or directly from Heroku audio stream.
4. **Summarization**
    - Selected text sent to Heroku `/summarize` endpoint.
    - Response returned to desktop app and cached locally.
5. **Persistence**
    - SQLite DB stores:

        - file paths
        - read progress
        - bookmarks & annotations
        - TTS cache locations

---

5. Software Stack

Local (Desktop)

Category	Tech Stack
GUI	Electron + React + TailwindCSS
Local API	FastAPI (Python 3.12)
Parsing	`pdfplumber`, `ebooklib`, `PyMuPDF`, `docx2txt`
TTS (optional local)	Coqui-TTS
Storage	SQLite
Audio Playback	PyAudio or HTML5 Audio via Electron
Environment Management	Poetry / Pipenv
Packaging	Electron Builder

Heroku (Cloud)

Layer	Tech
Web Server	FastAPI / Flask
Task Queue	Celery + Redis (for longer TTS jobs)
AI Libraries	OpenAI API, HuggingFace Transformers, Whisper
Storage	Local ephemeral or S3-compatible bucket

Deployment	Heroku Pipelines (Staging / Prod)
Security	API key authentication (personal key only)

---

6. Local–Cloud Integration

Local Endpoint:

```
POST /api/tts
{
  "text": "...",
  "voice": "rich-voice-1",
  "mode": "cloud" | "local"
}
```

Cloud Endpoint (Heroku):

```
POST /api/v1/tts
{
  "text": "...",
  "model": "gpt-4o-tts",
  "voice": "alloy"
}
```

Heroku Returns:

- Streamed or pre-generated `.mp3`
- Response metadata (`duration`, `sample_rate`, `voice_used`)

---

7. Security Model

- Local FastAPI only binds to `localhost:5000` (not exposed publicly).

- All cloud interactions use HTTPS with personal bearer token.
- No third-party user accounts or telemetry.
- Heroku stores no user data — ephemeral compute only.

---

8. Performance and Scalability

Operation	Location	Target Performance
Parsing a 200-page PDF	Local	< 5 seconds
Local TTS (Coqui)	Local	1x real-time
Cloud TTS (Heroku)	Remote	< 2 sec latency
Summarization (GPT-4o)	Remote	< 8 sec per chapter
DB read/write	Local SQLite	instantaneous

---

9. Deployment & DevOps

Area	Tool
Local App Packaging	Electron Builder (creates `.app` / `.exe`)
Backend Hosting	Heroku Standard Dyno
Database Migration	Alembic (for Heroku Postgres if used)
Version Control	Git + GitHub
CI/CD	GitHub Actions → Auto-deploy to Heroku
Local Runtime	Docker Compose (optional)
Logs & Monitoring	Heroku Logs + Sentry (local + cloud)

---

10. Roadmap

Phase	Deliverables
Phase 1 (MVP)	Electron app + local FastAPI + PDF/EPUB parsing + basic playback
Phase 2	Integrate Heroku TTS + caching + voice selector
Phase 3	Add summarization endpoint + local note system
Phase 4	Coqui-TTS local fallback + offline speech synthesis
Phase 5	UI polish + exportable audio + hotkey navigation

---

11. Directory Layout (Proposed)

```
readme-app/
├── frontend/                # Electron + React
│   ├── src/
│   │   ├── components/
│   │   ├── pages/
│   │   └── store/
│   └── package.json
├── backend/
│   ├── main.py              # FastAPI local API
│   ├── tts/
│   │   ├── coqui.py
│   │   ├── openai_cloud.py
│   └── parsers/
│       ├── pdf_parser.py
│       ├── epub_parser.py
│       └── doc_parser.py
├── cloud/
│   ├── app.py               # Heroku FastAPI entry
│   ├── summarize.py
│   ├── tts_openai.py
│   └── requirements.txt
├── db/
│   └── readme.db
└── config/
    ├── settings.yaml
    └── secrets.env
```

---

12. Technologies Summary

Layer	Stack
Frontend	Electron + React + TailwindCSS
Backend (Local)	FastAPI, PyMuPDF, pdfplumber, Coqui-TTS, SQLite
Backend (Heroku)	FastAPI, OpenAI API, Celery, Redis
Infrastructure	Docker, GitHub Actions, Heroku Dynos
Security	API key, localhost-only exposure, TLS
Audio Pipeline	Local caching, async streaming

Technical Specification (TS): “ReadMe” – Local AI Reading Assistant

---

1. System Topology

Architecture Overview

```
┌──────────────────────────────────────┐
 │             Desktop Layer             │
 │  Electron + React (UI)                │
 │  communicates via REST/WebSocket      │
 └──────────────────────────────────────┘
                  │
           localhost:5000
                  │
 ┌──────────────────────────────────────┐
 │           Local Backend               │
 │  FastAPI + SQLite + Coqui-TTS         │
 │  Parsing + Caching + Playback Mgmt    │
```

```
└─────────────────────────────────────┘
                  │ (secure HTTPS)
                  ▼
┌─────────────────────────────────────┐
│            Cloud Compute            │
│  Heroku Dyno (FastAPI / Celery)     │
│  OpenAI TTS / GPT Summarizer        │
└─────────────────────────────────────┘
```

---

2. Core Data Structures

2.1 Book Metadata

```
class Book(BaseModel):
    id: str                     # UUID
    title: str
    author: Optional[str]
    filepath: str               # Absolute local path
    filetype: str               # pdf | epub | txt | docx
    total_pages: int
    last_read_page: int = 0
    last_accessed: datetime
    cover_image_path: Optional[str]
    text_cache_path: Optional[str]
```

2.2 Annotation / Note

```
class Annotation(BaseModel):
    id: str
    book_id: str
    page_number: int
    selection_text: str
    note_text: str
    timestamp: datetime
```

2.3 Audio Job (Local/Remote)

```
class AudioJob(BaseModel):
    id: str
    book_id: str
    chapter: str
    status: str                 # queued | processing | done | failed
    engine: str                 # local | cloud
    voice: str
    text_path: str
    audio_path: Optional[str]
    created_at: datetime
```

---

3. Database Schema (SQLite)

```
CREATE TABLE books (
    id TEXT PRIMARY KEY,
    title TEXT,
    author TEXT,
    filepath TEXT,
    filetype TEXT,
    total_pages INTEGER,
    last_read_page INTEGER,
    last_accessed DATETIME,
    cover_image_path TEXT,
    text_cache_path TEXT
);

CREATE TABLE annotations (
    id TEXT PRIMARY KEY,
    book_id TEXT,
    page_number INTEGER,
    selection_text TEXT,
    note_text TEXT,
    timestamp DATETIME,
    FOREIGN KEY(book_id) REFERENCES books(id)
);

CREATE TABLE audio_jobs (
    id TEXT PRIMARY KEY,
    book_id TEXT,
    chapter TEXT,
    status TEXT,
    engine TEXT,
    voice TEXT,
    text_path TEXT,
    audio_path TEXT,
    created_at DATETIME,
    FOREIGN KEY(book_id) REFERENCES books(id)
);
```

---

4. API Specification

4.1 Local FastAPI (localhost:5000)

Method	Endpoint	Description
`POST`	`/api/books/import`	Upload & parse `.pdf`, `.epub`, etc. Returns metadata.
`GET`	`/api/books`	List all local books with metadata.
`GET`	`/api/books/{id}`	Get details, progress, and cached content.
`POST`	`/api/books/{id}/parse`	Force re-parse a file.
`POST`	`/api/tts`	Generate audio (mode: `local` or `cloud`).
`GET`	`/api/audio/{id}/stream`	Stream cached `.wav`/`.mp3` to player.
`POST`	`/api/annotate`	Add a note/bookmark.
`GET`	`/api/annotations/{book_id}`	Retrieve annotations.
`POST`	`/api/summarize`	Send selected text to Heroku summarization service.
`GET`	`/api/settings`	Retrieve system config (voices, cache paths).

`PUT`	`/api/settings`	Update local preferences.

4.2 Cloud Heroku FastAPI (https://readme-ai.herokuapp.com)

Method	Endpoint	Description
`POST`	`/api/v1/tts`	Convert text → audio using OpenAI TTS or another engine.
`POST`	`/api/v1/summarize`	Summarize a paragraph or chapter.
`POST`	`/api/v1/translate`	(Optional) Translate selected text.
`GET`	`/api/v1/health`	Health check.

4.3 Example Payloads

Local → Cloud TTS Request

```
POST /api/v1/tts
{
  "text": "Chapter one begins...",
  "model": "gpt-4o-tts",
  "voice": "alloy"
}
```

Response

```
{
  "job_id": "a2b1f...",
  "duration": 32.6,
  "sample_rate": 44100,
  "audio_url": "https://readme-ai.herokuapp.com/files/a2b1f.mp3"
}
```

---

5. Processing Pipelines

5.1 File Parsing (Local)

1. Electron selects file → POST `/api/books/import`
2. FastAPI determines parser based on file extension

Extracted text stored in JSON cache:

```
cache/books/<book_id>/text.json
```

3.
4. DB entry created.

5.2 Text-to-Speech

- **Local Mode:** Coqui-TTS engine generates audio to cache directory.
- **Cloud Mode:** Text sent to Heroku → processed via OpenAI API → audio URL returned → downloaded + cached locally.

5.3 Summarization

1. Electron highlights passage → POST `/api/summarize`
2. FastAPI forwards to Heroku `/api/v1/summarize`
3. GPT-4o generates abstract → result cached in SQLite.

---

6. Heroku Cloud Services

6.1 Heroku App Structure

```
cloud/
├── app.py
├── routers/
```

```
│   ├── tts.py
│   └── summarize.py
├── services/
│   ├── openai_client.py
│   └── audio_utils.py
└── Procfile
```

6.2 Example Heroku Procfile

```
web: gunicorn app:app --workers=1 --timeout 120
worker: celery -A tasks.celery_app worker --loglevel=info
```

6.3 Celery Task Example

```
@app.task
def generate_audio_task(text, voice="alloy"):
    audio = openai.audio.speech.create(
        model="gpt-4o-mini-tts",
        voice=voice,
        input=text
    )
    filename = f"{uuid.uuid4()}.mp3"
    with open(f"/tmp/{filename}", "wb") as f:
        f.write(audio.read())
    return f"/tmp/{filename}"
```

---

7. Desktop Application (Electron + React)

7.1 UI Layout

Component	Purpose
Sidebar	Library (Book list, progress bar)
Reader Pane	Paginated text view, highlighting, annotations
Player Bar	Play/pause, speed, seek, voice selector

Settings Panel	Choose local vs cloud mode, manage voices
Console Panel (Dev)	Display FastAPI logs, status updates

7.2 IPC Communication

Electron bridges frontend ↔ Python backend:

```
ipcRenderer.invoke('fetch-books')
ipcRenderer.invoke('start-tts', { text, mode: 'local' })
ipcRenderer.invoke('summarize-text', { text })
```

Electron executes Python server using:

```
const pyProc = spawn('python', ['backend/main.py'])
```

---

8. Data Flow Diagram

```
┌────────────────┐        ┌──────────────┐        ┌─────────────┐
│   Electron UI  │  ───▶  │ Local FastAPI│  ───▶  │ Heroku Cloud│
└────────────────┘        └──────────────┘        └─────────────┘
       ▲                         │                        │
       │                         │                        │
       │                         ▼                        ▼
       │                  Coqui-TTS / SQLite       OpenAI API / GPT-4o
       │
       └── Playback, Notes, and Rendering
```

---

9. Configuration and Secrets

```
config/settings.yaml
```

```
tts_default: "local"
voices:
  - "coqui_en"
  - "openai_alloy"
heroku_api_url: "https://readme-ai.herokuapp.com"
api_key: !ENV ${HEROKU_API_KEY}
cache_dir: "./cache"
```

```
config/secrets.env
```

```
OPENAI_API_KEY=sk-...
HEROKU_API_KEY=...
```

---

10. Performance Targets

Function	Target
PDF parse (200 pages)	< 5 sec
Local TTS latency	< 1.5x realtime
Cloud TTS round-trip	< 2 sec
Summarization response	< 8 sec
App cold start	< 3 sec
Memory footprint	< 600 MB

---

11. Error Handling and Recovery

Scenario	Mitigation
Heroku offline	Fallback to local TTS
Parsing error	Retry with alternate parser
API timeout	Queue and retry async
Disk full	Alert and clear cache
DB locked	Rollback + exponential backoff

---

12. Testing Strategy

Level	Tools	Purpose
Unit Tests	Pytest, Jest	Core parsing, API endpoints
Integration	Postman, pytest-asyncio	Local ↔ Heroku endpoints
UI Tests	Cypress	GUI behavior
Load Tests	Locust	Parsing + TTS concurrency
E2E	Electron Test Kit	Full pipeline validation

---

13. Deployment Guide Summary

Local Setup

```
 git clone readme-app
cd backend && poetry install
poetry run uvicorn main:app --reload
cd ../frontend && npm install && npm run electron-dev
```

1.

Heroku Setup

```
 heroku create readme-ai
git push heroku main
heroku config:set OPENAI_API_KEY=sk-...
heroku ps:scale web=1 worker=1
```

2.

---

14. Security Summary

- **Localhost-only** backend access.

- **Cloud auth** via static bearer key (personal).
- **Encrypted storage** for annotations and cache (SQLCipher optional).
- **All external calls HTTPS-only**.
- **No analytics / telemetry / external tracking**.

---

15. Optional Enhancements

Feature	Description
Voice Cloning	Train voice using Coqui-VC and store locally.
Web Scraper	Import articles directly from URLs.
AI Q&A Mode	Ask contextual questions about chapters using GPT-4o.
Batch Export	Generate audiobook MP3 for entire book.
CLI Interface	Lightweight terminal mode for TTS playback.

Cloud repo layout

```
readme-cloud/
├─ app.py
├─ settings.py
├─ auth.py
├─ routers/
│   ├─ tts.py
│   └─ summarize.py
├─ services/
│   ├─ openai_tts.py
│   └─ openai_summarize.py
├─ tests/
│   ├─ test_tts.py
│   └─ test_summarize.py
```

```
├─ requirements.txt
├─ Procfile
└─ runtime.txt
```

Environment variables (Heroku)

```
OPENAI_API_KEY=...
API_BEARER=some-long-random-token             # your personal bearer
for cloud auth
AUDIO_FORMAT=mp3                              # mp3 | wav | ogg
(optional)
OPENAI_TTS_MODEL=gpt-4o-mini-tts              # default TTS model
OPENAI_SUMMARY_MODEL=gpt-4.1-mini             # or another lightweight
text model
```

Notes:

- The **Audio API** `audio/speech` endpoint with models like `gpt-4o-mini-tts` is the current OpenAI path for TTS. OpenAI Platform+2OpenAI Platform+2
- Keep an eye on OpenAI's changelog/deprecations and the newer **Responses API** guidance for text to avoid breaking changes. OpenAI Platform+2OpenAI Platform+2

`requirements.txt`

```
fastapi==0.115.0
uvicorn==0.30.6
gunicorn==23.0.0
pydantic==2.9.2
httpx==0.27.2
python-multipart==0.0.9
```

(The official `openai` SDK evolves; this skeleton uses `httpx` against stable REST endpoints.)

`Procfile`

```
web: gunicorn app:app --workers=1 --timeout=120
```

`runtime.txt`

```
python-3.12.6
```

`settings.py`

```
from pydantic import BaseSettings, Field

class Settings(BaseSettings):
    openai_api_key: str = Field(alias="OPENAI_API_KEY")
    api_bearer: str = Field(alias="API_BEARER")
    audio_format: str = Field(default="mp3", alias="AUDIO_FORMAT")
    openai_tts_model: str = Field(default="gpt-4o-mini-tts",
alias="OPENAI_TTS_MODEL")
    openai_summary_model: str = Field(default="gpt-4.1-mini",
alias="OPENAI_SUMMARY_MODEL")

    class Config:
        env_file = ".env"
        case_sensitive = True

settings = Settings()
```

`auth.py` (simple bearer auth)

```
from fastapi import Header, HTTPException, status
from settings import settings

def verify_bearer(authorization: str = Header(...,
convert_underscores=False)):
    # Expect: Authorization: Bearer <token>
    if not authorization or not
authorization.lower().startswith("bearer "):
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Missing bearer token")
    token = authorization.split(" ", 1)[1].strip()
    if token != settings.api_bearer:
        raise HTTPException(status_code=status.HTTP_403_FORBIDDEN,
detail="Invalid token")
    return True
```

`services/openai_tts.py`

```
import httpx
from typing import AsyncIterator
from settings import settings

# OpenAI Text-to-Speech via Audio API (audio/speech)
# Docs: Audio & TTS overview + models (e.g., gpt-4o-mini-tts).
# We'll stream bytes and forward as a generator.
# References: Audio & speech docs; Text-to-speech guide; Model page.
# (Citations in the main response.)
OPENAI_TTS_ENDPOINT = "https://api.openai.com/v1/audio/speech"

async def tts_stream(text: str, voice: str | None = None,
audio_format: str | None = None) -> AsyncIterator[bytes]:
    payload = {
        "model": settings.openai_tts_model,
        "input": text,
        # "stream": True   # If/when OpenAI supports streaming
responses for TTS in this endpoint.
        "voice": voice or "alloy",
        "format": (audio_format or settings.audio_format),
    }

    headers = {
        "Authorization": f"Bearer {settings.openai_api_key}",
        "Content-Type": "application/json",
    }

    async with httpx.AsyncClient(timeout=None) as client:
        # We use stream=True to forward the response as it arrives.
        async with client.stream("POST", OPENAI_TTS_ENDPOINT,
headers=headers, json=payload) as resp:
            resp.raise_for_status()
            async for chunk in resp.aiter_bytes():
                if chunk:
                    yield chunk
```

---

services/openai_summarize.py

```
import httpx
from settings import settings

# For summarization we prefer the newer Responses API semantics as
they roll out.
# Until fully standardized, this uses the /chat/completions-equivalent
JSON.
# If you migrate to the Responses API, change ENDPOINT & payload shape
accordingly.

OPENAI_TEXT_ENDPOINT = "https://api.openai.com/v1/chat/completions"

SUMMARY_SYS = (
    "You are a concise academic summarizer. "
    "Write a faithful, non-speculative abstract of the input text. "
    "Return 3-7 sentences, preserve key terms, and avoid quotations."
)

async def summarize(text: str, max_tokens: int = 300) -> str:
    headers = {
        "Authorization": f"Bearer {settings.openai_api_key}",
        "Content-Type": "application/json",
    }
    payload = {
        "model": settings.openai_summary_model,
        "messages": [
            {"role": "system", "content": SUMMARY_SYS},
            {"role": "user", "content": text},
        ],
        "temperature": 0.2,
        "max_tokens": max_tokens,
    }

    async with httpx.AsyncClient(timeout=120) as client:
        r = await client.post(OPENAI_TEXT_ENDPOINT, headers=headers,
json=payload)
        r.raise_for_status()
        data = r.json()
        return data["choices"][0]["message"]["content"].strip()
```

*(If you adopt the **Responses API** later, switch to its request/streaming format per docs. OpenAI Platform)*

---

routers/tts.py

```
from fastapi import APIRouter, Depends, HTTPException, status
from fastapi.responses import StreamingResponse
from pydantic import BaseModel, Field
from typing import Optional
from auth import verify_bearer
from services.openai_tts import tts_stream
from settings import settings

router = APIRouter(prefix="/api/v1", tags=["tts"])

class TTSIn(BaseModel):
    text: str = Field(min_length=1, max_length=200_000)
    voice: Optional[str] = None
    format: Optional[str] = None

@router.post("/tts")
async def tts(req: TTSIn, _=Depends(verify_bearer)):
    if not req.text.strip():
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail="Text is empty")
    fmt = (req.format or settings.audio_format).lower()
    media_type = {
        "mp3": "audio/mpeg",
        "wav": "audio/wav",
        "ogg": "audio/ogg",
    }.get(fmt, "audio/mpeg")

    gen = tts_stream(text=req.text, voice=req.voice, audio_format=fmt)
    return StreamingResponse(gen, media_type=media_type)
```

---

routers/summarize.py

```
from fastapi import APIRouter, Depends, HTTPException, status
from pydantic import BaseModel, Field
from auth import verify_bearer
from services.openai_summarize import summarize

router = APIRouter(prefix="/api/v1", tags=["summarize"])

class SummarizeIn(BaseModel):
    text: str = Field(min_length=1, max_length=200_000)
    max_tokens: int = Field(default=300, ge=64, le=2000)

class SummarizeOut(BaseModel):
    summary: str

@router.post("/summarize", response_model=SummarizeOut)
async def summarize_endpoint(req: SummarizeIn,
_=Depends(verify_bearer)):
    if not req.text.strip():
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail="Text is empty")
    out = await summarize(req.text, max_tokens=req.max_tokens)
    return SummarizeOut(summary=out)
```

`app.py`

```
from fastapi import FastAPI
from routers import tts, summarize

app = FastAPI(title="ReadMe Cloud API", version="0.1.0")
app.include_router(tts.router)
app.include_router(summarize.router)

@app.get("/api/v1/health")
async def health():
    return {"ok": True}
```

Minimal tests (pytest-style)

`tests/test_tts.py`

```
import os
```

```
from httpx import AsyncClient
from app import app

API_BEARER = os.getenv("API_BEARER", "test-token")

async def test_tts_health():
    async with AsyncClient(app=app, base_url="http://test") as ac:
        r = await ac.get("/api/v1/health")
        assert r.status_code == 200

async def test_tts_requires_auth():
    async with AsyncClient(app=app, base_url="http://test") as ac:
        r = await ac.post("/api/v1/tts", json={"text": "Hello"})
        assert r.status_code == 401
```

```
tests/test_summarize.py
```

```
import os
from httpx import AsyncClient
from app import app

API_BEARER = os.getenv("API_BEARER", "test-token")

async def test_summarize_requires_auth():
    async with AsyncClient(app=app, base_url="http://test") as ac:
        r = await ac.post("/api/v1/summarize", json={"text": "x"*100})
        assert r.status_code == 401
```

Example calls from your local app

TTS (streamed MP3):

```
curl -X POST https://<your-heroku-app>.herokuapp.com/api/v1/tts \
  -H "Authorization: Bearer $API_BEARER" \
  -H "Content-Type: application/json" \
  --data '{"text":"Chapter one begins...", "voice":"alloy",
"format":"mp3"}' \
  --output chapter1.mp3
```

Summarize:

```
curl -X POST https://<your-heroku-app>.herokuapp.com/api/v1/summarize
\
  -H "Authorization: Bearer $API_BEARER" \
  -H "Content-Type: application/json" \
  --data '{"text":"<long passage here>", "max_tokens": 300}'
```

Security and durability notes

- Endpoints require your **personal bearer**; rotate `API_BEARER` regularly.
- All OpenAI calls are HTTPS and scoped by `OPENAI_API_KEY`.
- No files persist on Heroku; you stream bytes directly back to the desktop client.
- If you later add job queues or pre-generation, introduce a signed, time-limited URL layer.

Where this aligns with current OpenAI docs

- **Text-to-Speech**: Audio API “audio/speech” with models incl. `gpt-4o-mini-tts`; guide and model pages outline inputs and voice options. OpenAI Platform+2OpenAI Platform+2
- **Model/version hygiene**: Monitor **Changelog** and **Deprecations**; if you switch to the **Responses API** for text, adjust the summarization service accordingly. OpenAI Platform+2OpenAI Platform+2