✅ COMPLETE PROJECT PLAN: STORY2AUDIO MICROSERVICE

🎯 **Objective**

Build a self-contained microservice that:

- **Converts text stories to emotionally engaging audio**
- **Has a gRPC API**
- **Supports concurrency, error handling**
- **Provides test cases, frontend demo, and full documentation**
- **Is deployable via Docker and easily reproducible**

PHASE 1: Project Initialization & Setup

🔹 Step 1: Set Up GitHub Repository

https://github.com/Albab001/-Story2Audio-Microservice.git

- Create a private/public repo on GitHub titled: `Story2Audio-Microservice`

Setup folder structure:

```
css
CopyEdit
├── src/
├── models/
├── api/
├── frontend/
├── tests/
├── Dockerfile
├── README.md
└── requirements.txt
```

-

🔹 Step 2: Environment Setup

- Set up a Python virtual environment.

Install basic tools:

bash
CopyEdit

```
pip install grpcio grpcio-tools fastapi uvicorn torch torchaudio transformers pydantic
```

-
- Optional (for frontend later): `pip install gradio streamlit`

PHASE 2: Model Selection and Integration

🔹 Step 3: Choose Core NLP and TTS Models

You **should not use cloud APIs** — models must run locally.

- For **story enhancement / natural storytelling tone**:
    - Use: `phi-2` (or `LLaMA2`/`Mistral-7B` if RAM allows)
- For **Text-to-Speech with emotion**:
    - Use: `Bark by Suno` or `Coqui TTS` (Both are open-source and support emotion/tone variation)

🔹 Step 4: Integrate TTS and Story Models

- Write a wrapper for the language model (LLaMA/phi-2) to enhance user-provided story.
- Feed enhanced story to `Bark`/`Coqui TTS` and generate audio.
- Store audio file temporarily or return as base64 blob.

PHASE 3: API Development (gRPC Focus)

🔹 Step 5: Define gRPC Protobuf Contracts

Create `story2audio.proto` with:

proto
CopyEdit

```
service StoryService {
  rpc GenerateAudio (StoryRequest) returns (AudioResponse) {}
}

message StoryRequest {
  string story_text = 1;
}

message AudioResponse {
  string status = 1;
  string audio_base64 = 2;
  string message = 3;
}
```

-

🔹 Step 6: Implement gRPC Server with Async

- Implement server logic in `api/` to handle:
    - Receiving input
    - Validating input
    - Enhancing and converting text → audio
    - Returning base64 audio with appropriate status codes

🔹 Step 7: Add Error Handling

- Check for:
    - Empty string input
    - Audio generation failure
    - Timeouts or concurrency overloads
- Return proper JSON messages & codes

PHASE 4: Frontend & Postman Testing

🔹 Step 8: Build Frontend UI

- Use **Gradio** (or Streamlit) to demo:
    - Text input box for story
    - Audio output playback
    - Status messages
- Keep frontend independent; it should call your gRPC API via Python client

🔹 Step 9: Create Postman Collection

- Use `grpc` extension or HTTP wrapper to test core functionality
- Write 5–10 test cases:
    - Normal story
    - Very long story
    - Empty string
    - Gibberish

    - Concurrent requests

PHASE 5: Testing & Evaluation

🔹 Step 10: Functional & Performance Testing

- Write unit and integration tests using `pytest`
- Simulate concurrent requests (e.g., using `locust`, `ab`, or Python threads)
- Plot:
    - Response time vs. # of users
    - Success/failure rates

🔹 Step 11: Prepare Test Cases & Reports

- Create `tests/` folder with:
    - Input stories (JSON)
    - Expected outputs (base64/audio, logs)
    - README for running tests

PHASE 6: Deployment & Documentation

🔹 Step 12: Containerization

- Write `Dockerfile`:
    - Base image: Python

    - Install all requirements
    - Start gRPC server on entrypoint

🔹 Step 13: Deployment Script

- Create `docker-compose.yml` (optional)
- Add bash script: `deploy.sh` to build and run the container

🔹 Step 14: Write Full Documentation

- `README.md` should include:
    - Project overview
    - Setup instructions
    - API details + gRPC contract
    - Model info (source + citation)
    - Limitations and challenges
    - How to run tests
    - Architecture diagram

PHASE 7: Final Delivery

🔹 Step 15: Video Demo (Optional but Recommended)

- Record a 1–2 minute video showing:
    - Gradio app working
    - gRPC call in Postman

- Test cases
- Performance plots

🔹 Step 16: Submission Checklist

- ✅ Source Code in GitHub
- ✅ Gradio Frontend
- ✅ Postman Collection
- ✅ Dockerfile & deployment
- ✅ Full documentation (README)
- ✅ Testcases + Performance Evaluation
- ✅ (Optional) Video Demo

📌 Technologies Summary

Purpose	Tools/Models
TTS	Bark / Coqui TTS
Text enhancement	Phi-2 / LLaMA2
API	gRPC + Python (asyncio)
Frontend	Gradio or Streamlit
Testing	Postman, pytest
Deployment	Docker
Performance Eval	locust/ab, matplotlib