✅ Phase 1: Initial Setup & Planning

1.1 Define Microservice Goals

- **Input:** A story in plain text (500–1500 words).
- **Output:** Narrated audio file (.mp3 or .wav) + optional JSON metadata.
- **Tech Stack (suggested):**
    - Language: Python (FastAPI for backend).
    - Model: Bark (by Suno) or Coqui TTS (local), not cloud APIs.
    - TTS Preprocessing: LLM (GPT-4, DeepSeek) for splitting text into expressive sentences/dialogues.

---

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

✅ Phase 2: Microservice Core Functionality

2.1 Preprocess the Story

- **What to do:** Break the story into narratable chunks (~150 words per chunk).
- **Why:** TTS models can't handle large text at once.
- **How:** Use GPT-4 locally (or DeepSeek/Cursor) to generate `chunked_text = [para1, para2, ...]`.

2.2 TTS Model Selection & Setup

- **Option 1: Bark (offline)**:
    - High-quality expressive voice.
    - Can be run locally with some setup.
- **Option 2: Coqui TTS**:

- Lighter and easier to run.
- Use a pretrained expressive model (like `tts_models/multilingual/multi-dataset/your_tts`).

👉 **Install and test your chosen model locally.**

2.3 Implement Core Audio Generation

- **Loop through `chunked_text[]`**, convert each to audio.
- **Concatenate** or **stitch** audio files (use `pydub`).
- Return final output as `.wav` or `.mp3`.

✅ Phase 3: Build the API

3.1 Backend: FastAPI Setup

- Create a `/generate-audio` endpoint.
- Accept `POST` requests with JSON:

json
CopyEdit

```
{
  "story": "Once upon a time..."
}
```

- Respond with:

json
CopyEdit

```
{
  "status": "success",
  "audio_file_url": "/audio/story123.mp3"
```

```
}
```

3.2 Add Error Handling

- Handle empty input, long text errors, TTS model crashes.
- Return proper HTTP status codes and messages.

3.3 Add Async Support

- Use Python `async` features to make your API responsive.
- Especially useful when generating audio in chunks.

✅ Phase 4: Testing & Coverage

4.1 Auto-generate Test Cases

- Use GPT-4/DeepSeek to create:
    - Long story input
    - Dialogue-rich story
    - Invalid input (e.g., empty JSON, image instead of text)

4.2 Postman Testing

- Create a Postman collection:
    - Add successful, boundary, and failure cases.
    - Export and attach to your project submission.

✅ Phase 5: Frontend & Deployment

5.1 Basic Frontend (Optional but Recommended)

- Use HTML + JavaScript or React.
- Form with text input + “Generate Audio” button.
- On submission, call the FastAPI endpoint and play audio.

5.2 Deployment

- Use **Render**, **Railway**, or **Docker + VPS** (like Fly.io or DigitalOcean).
- Ensure Bark or Coqui runs without external cloud dependencies.
- Share public URL for demo.

✅ Phase 6: Documentation & Final Touches

6.1 Write Detailed README

- What it does, how to run, endpoints, model used.
- Add architecture diagram (use draw.io).

6.2 Add In-Code Docstrings

- Each function must have docstrings explaining purpose, inputs, outputs.

6.3 Performance Evaluation

- Mention model inference time, latency per request.
- Show memory/CPU usage if possible.

6.4 Presentation Slide Plan

- Problem statement
- Your solution (Story2Audio)
- Flow diagram (text → chunks → audio → stitched output)
- Models used
- API demo
- Challenges + what you learned

✅ Bonus: Optional Enhancements

- **Speaker Control:** Let user choose gender/mood of the voice.
- **Multi-language Support:** Coqui supports this.
- **Emotion Control:** Slightly advanced; Bark does this better.

🔚 Final Deliverables Checklist

Task	Status
[ ] TTS Model setup and tested locally	–
[ ] Preprocessing chunker using GPT-4	–
[ ] FastAPI microservice	–
[ ] Async & error handling	–
[ ] Postman collection	–
[ ] Frontend basic interface	–

[ ] Deployment live demo –

[ ] Documentation & slides –