VoiceOwl Developer Evaluation Task

Role: Node.js + TypeScript Developer
Focus: Backend development, code structure, MongoDB usage, API integrations, scalability, clarity of thinking

---

🚀 Task Summary

Build a minimal API service that accepts an audio file URL, mocks transcription, and stores the result in MongoDB.

If you're also applying as a full-stack developer, add a simple frontend UI for testing and viewing transcriptions.

Additionally, extend the service with MongoDB queries, scalability design, and an external API integration (Azure Speech).
Realtime/workflow tasks are optional for bonus points.

Part 1 – Backend API (Required)

Functionality

Implement an HTTP `POST /transcription` endpoint.

The request body will look like:

```
{

  "audioUrl": "https://example.com/sample.mp3"

}
```

The server should:

- Download the audio file from the given URL (**mock this step**).
- “Transcribe” it by returning a dummy transcription like `"transcribed text"`.

- Save `{ audioUrl, transcription, createdAt }` into MongoDB.
- Return the MongoDB record's `_id` in the API response.

🧱 Tech Requirements

- Node.js + TypeScript
- Express or Fastify
- MongoDB (local, Atlas, or MongoMemoryServer)
- Clean code structure (services, routes/controllers, models)
- Basic error handling

🧪 Bonus

- Use environment variables (dotenv)
- TypeScript interfaces for request/response types
- Include a test case using Jest or similar
- Implement a simple retry if the download fails

Part 2 – MongoDB Query & Indexing (Required)

Extend your service with:

- `GET /transcriptions` → fetch only those created in the **last 30 days**.

In your **README**, briefly explain:

- What **index** you would add for this query if the dataset had 100M+ records.

Part 3 – Scalability & System Design (Required)

In your **README**:

- Describe how you'd evolve your service to handle **10k+ concurrent requests**.
- Mention 2–3 changes (e.g., caching, queues, containerization, autoscaling).
- Keep it short and practical — no detailed diagrams required.

Part 4 – API Integration (Required)

Add integration with **Azure Speech-to-Text** (or mock if credentials unavailable).

- Implement `POST /azure-transcription` endpoint:

Request body:

```
{ "audioUrl": "https://example.com/sample.mp3" }
```

  -
  - Mock download audio.
  - Use Azure Cognitive Services Speech SDK to transcribe (or stub if no key).
  - Save `{ audioUrl, transcription, source: "azure", createdAt }` in MongoDB.
  - Handle API keys via environment variables.
  - Gracefully handle API errors/timeouts.

🧪 Bonus

- Support multiple languages (e.g., `en-US`, `fr-FR`).
- Retry failed requests with exponential backoff.

Part 5 – Realtime / Workflow (Optional, Bonus)

Choose **one**:

Option A – Realtime Voice Streaming

- Add a WebSocket endpoint that accepts mocked audio chunks.
- Stream back dummy transcription events (e.g., `{ partial: "…" }`).
- Store metadata in Mongo.

Option B – Workflow Engine

- Implement a simple workflow: transcription → review → approval.
- Steps should progress asynchronously.
- Persist workflow state in MongoDB.

Part 6 – Frontend (Optional for Full Stack)

🌐 [Optional] Extend your project with a frontend that:

- Accepts user input for `audioUrl`
- Sends it to `POST /transcription`
- Shows the result (transcription ID and message)
- (Optional) Lists all transcriptions from `GET /transcriptions`

💻 Tech Expectations

- React or Next.js
- TypeScript
- Organized structure (components, services, etc.)
- API handling with Axios, React Query, or fetch
- Basic styling (Tailwind, CSS, or Chakra)

📦 Submission Instructions

Please share the following:

- **Codebase**: GitHub repo or downloadable ZIP
- **README.md** with:
  - Explanation of code structure
  - Assumptions made
  - How you'd improve it for production
  - MongoDB indexing notes
  - Scalability notes
- **Loom/screen recording (2–5 mins)** walking through the code

🧑‍💻 If You Built the Frontend

- Place it in `/client` or `/frontend`
- Add a README section for frontend
- Loom walkthrough showing both backend + frontend working