Web/xQuery project: Interview database

I'm starting to work with interviews. I would like to be able to get the audio transcribed to text and then store the text and the metadata about the interview in an no-sql/non-relational database that I can then query using either xQuery, NoSQL or SQL databases.

The idea:

> Create a tool to automatically transcribe interviews and other audio files into an XML database. Then use xQuery to build the metadata for the interviews.

These are preliminary notes and ideas, where possible I will try to incorporate code to validate them but it won't be perfect or possible to do everywhere. Future posts will update progress on the different parts of the project.

Tooling and processes

There are two or three technologies that would make the project as envisioned possible:

- A noSQL or xQuery database
- A storage bucket for the audio files
    - This depend on what database and speech to text technologies I choose
- A speech to text engine/technology
- A server to run the content from
    - This is also dependent on the xQuery or noSQL database we choose

Speech to text options

Product	Vendor	Pricing URL	Free Tier	Notes
Google speech-to-text	Google	Pricing		

Product	Vendor	Pricing URL	Free Tier	Notes
Microsoft speech-to-text	Microsoft	Pricing		
Amazon Transcribe	Amazon	Pricing		
Deep Speech	Mozilla	Open Source	N/A	Released under the Mozilla Public License 2.0 Also see Use Mozilla DeepSpeech to enable speech to text in your application Python only, doesn't appear to have a javascript or Node version or wrapper

I realize that it won't be long before the APIs start incurring cost, but as an experiment any of the APIs would work.

Identifying database options

The other area worth researching is whether xQuery is the right solution for this type of project. Other XML/noSQL databases are also a possibility.

Vendor	Type	License	Notes
eXist	NoSQL	Open Source	
MarkLogic	NoSQL	Commercial	
MongoDB	NoSQL	Varies	
PostrgreSQL	SQL	Open Source	

Even if xQuery is the right solution, then what is the best server to work with and how do we store the data? How expensive is it to host such a development solution in the cloud (either on premise or in the vendor's cloud)?

Creating a JSON schema for the data

JSON Schemas allows us to create a schema for JSON-based data. This ensures two things:

- Provides a way to validate the data

- Ensures data completeness and accuracy

I’ve validated the schema below using the [JSON Schema Validator]

The Schema

I’ve broken the schema into sections to make it easier to annotate and comment the different sections.

We first provide metadata about the schema. We give it a name,indicate what version of the JSON Schema specification we are using and specify the type for our root element.

```
{
  "title": "JSON schem"JSON schema for interviews""$schema"s://json-schem
  "type": "obje": ": "object",
```

We then start listing the schema properties. Most of the properties will have a type (what kind of value we want the property to have) and a human-readable description.

There are exceptions that will be documented where they appear.

```
  "properties": {
    "$schema": {
      "type": "string"
    },
    "string""id"  "type": "strin": "      "description": "Unique id": ""de
    },
    "title": {
      "type": "str": ""title": {
      "type""Title of the interview"
    },"Title of the interview": "Title of the interview"
    },
```

The type object deserves special mention as it deviates from our standard properties. It is meant to have one of a fixed set of values rather than a string we type in.

We set both a `default` value and an `enum` with the list of possible values.

```
    "type": {
      "type": "string",
      "string""description": "Type of interview",
      "default": "interview",
      "enum"    "interview",
        "interview-one-on-one",
        "interview-group",
        "interview-one-on-one-video",
        "interview-g"interview",
        "interview-one-on-one-audio",
        "interview-group-audio"
      ]
    },
```

`date` is a string formated as a date using the `yyyy-mm-dd` format. A valid example: `2018-11-13`

`location` is just a regular string.

```
    "date": {
      "type": "string",
      "string""description"of the interview",
      "format": "date"": " },
    "location": "location""location": {
      "type"cription": "Locati": ""description": "Location of the intervi
    },
```

`interviewers` and `interviewees` allow between one and five people to be listed.

The definition of a person is located under `$defs`. We define it outside the schema so we can use it in multiple locations.

```
    "interviewers": {
      "type": "array",
    "array""description": "Interviewer(s) of the interview",
    "minItems": 1,
    "maxItems": 5,
    "items"     "$ref": "#/$d": "person"
    }
  },
  "interviewees": {
    ""interviewees": {
    "type"tion": "Subject(s": ""description": "Subject(s) of the interv
    "minItems": 1,
    "maxItems" "$ref": "#": ""items": {
      "$ref": "#/$defs/person"
    }
  },
```

Audio is a nested object that has all the components that represent an audio file on the database. It also lists which children, if any, are required.

```
  "audio": {
    "type": "object",
    "object""properties": {
      "format"      "type": "strin": "          "description": "Format o
      "url": {
        "type": "s": ""url": {
        "type"n": "URL of the audio ": ""description"        "duration"
        "type": "numb": ""duration" "description": "Dur": ": "number",
        "description"  },
    "required": [
      "format",
      "url""format""required""required": [
      "format",
      "url",
      "duration"
    ]
  },
```

Likewise, the transcript is a collection of informatin about the transcription for the interview.

In this case, both the URL and the content are required.

```
    "transcript": {
      "type": "object",
      "object""properties": {
        "url"      "type": "strin": "          "description": "URL of th"
        },
        "content": {
          "type": "string": "         "description": "Content of the tr":
      "required": [
        "url",
        "content"
   "url""required": [
        "url",
        "content"
      ]
    }
```

The definitions in `@defs` can be used by reference elsewhere in the document.

For this schema the only value in `@defs` is `person`, which we use in `interviewers` and `interviewees`.

```
  },
  "$defs": {
    "person": {
      "type": "object",
      "object""properties": {
        "name"      "type": "strin": "          "description": "Name of t
        "email": {
          "type": "string": "         "description": "Email of the pers":
        "name"
      ]
    }
  }
}
"name""required": [
        "name"
        ]
      }
    }
  }
```

Now we have a schema to validate our data against it. It will also help in writing against the schema using an editor like VSCode or any of the IntelliJ IDEs.