Configuration guide

This configuration guide is intended for maintainers or solution architects who need to extend the corpus by adding new audio and metadata items.
It describes the required tools, assumptions, and processing steps needed to ensure consistency with the existing corpus. The guide is organized as a linear workflow: each section represents a step that should be completed in order, from source identification to final validation.

1. Required tools and environment

Ensure a Unix-like environment with a recent Python version, with the following python libraries and their dependencies properly installed and accessible as command-line tools:

- `yt-dlp` for audio extraction from online video sources
- `openai-whisper` for speech recognition and transcription
- `spacy` (along with an up-to-date italian model) for NLP and keywords extraction

The system also requires the universal media transcoder `ffmpeg` to be available both as `whisper` dependency and to later compress and optimize the audio file(s).
Given the potential computational intensity of most of these tasks, it might be recommended to use a high-performance cloud machine when processing many items or particularly long recordings.

2. Identify and acquire online source

Once the online video source has been identified proceed to the acquisition of its audio form via `yt-dlp`. While the simple tool invocation could work, it's better to leverage the existing best practice of appending the URL to the list of downloadables (in `.yt-dlp/barbero.lst`) and let the tool log its acquisition (to `.yt-dlp/barbero.log`), preventing duplication of data. The tool can then be invoked with all configuration flags via:

```
yt-dlp \
  --format bestaudio[ext=m4a] \
  --extract-audio --audio-format m4a \
  --sleep-interval 30 --limit-rate 5M \
  --batch-file .yt-dlp/barbero.lst \
  --download-archive .yt-dlp/barbero.log \
  --output "audio/barbero-%(extractor)s-%(id)s.%(ext)s"
```

We decided to use M4A format with AAC compression, as it provides better audio fidelity at smaller file sizes, which is beneficial for accurate transcription with `whisper`, compared to MP3 which introduces more compression artifacts that could hurt transcription accuracy.

3. Transcript audio file

Run `whisper` on the audio file using the italian language and `turbo` model, storing all transcription outputs in the transcripts directory. Do a quick quality check of the transcripts files: if for any reason the `turbo` model fails in transcribing some of the audio, retry using the `small` model:

```
whisper "your_file.m4a" \
  --model turbo \
  --language it \
  --task transcribe \
  --output_format all \
  --output_dir transcripts/
```

4. Extract keywords/entities and compile metadata chart

The keyword and entities extraction script uses a frequency analysis pipeline based on the most recurring words of the transcription text in order to perform a tentative NLP recognition of keywords (common nouns, verbs, etc…) and entities (persons, locations, etc…).
At the end of analysis, a JSON file is produced with 50 keywords and 30 entities that **must** be reviewed and selected manually for the specific entry, since the approach cannot assure precision, coherence, or deduplication.
Run the script on the transcription file via python interpreter

```
python scripts/keywords_extract.py transcriptions/your_transcription.txt
```

then, select the values on both sides to keep and append a proper metadata entry in the main `metadata.csv` file with all the required values, including the properly semantic-renamed filename.
For the entities then, run the authority control connector script in order to try to associate items to their wikipedia/wikidata/viaf authoritative reference.

```
python scripts/authority_connect.py \
  metadata/metadata.csv \
  metadata/entities-authoritative.csv
```

The results of this step too **must** be manually validated as NLP inference (and therefore subsequent associations) might be severely imprecise.

5. Optimize, validate and deliver

Now that the high-fidelity audio file has been transcribed and is no longer needed, it can be compressed in order to reduce the size and be served on the application efficiently. The chosen standard was AAC format with 48kbps bitrate, mono channel, and 22050Hz sample rate, which provides a good balance between audio quality and file size for spoken word content like lectures.
Run the audio file through `ffmpeg`

```
ffmpeg -i "audio/hq/your_audio.m4a" \
  -c:a aac -b:a 48k \
  -ac 1 -ar 22050 \
  "audio/compressed/compressed_audio.m4a"
```

Verify that audio, transcript, and metadata filenames match. Ensure metadata fields are complete, formats are consistent and contents are correct.
Upload the high quality file onto the AWS S3 bucket relative to the archival unit and the compressed file to the one relative to the application in production.
Finally, update the data JSON file that powers the web application by converting the `metadata.csv` with

```
python csv2json_convert.py metadata/metadata.csv html/metadata.json
```

verify the produced output is coherent and there haven’t been conversion errors, then commit the file onto the GitHub repository and wait for the automatic deploy via CI.
Confirm no errors are raised in the deployment and that the newly introduced entry shows and works properly on the website.