Adamas

↳ Web-application designed to allow users to mass produce audio books from pdf texts.

↳ 4 main features:

(i) Conventional Audiobook
(ii) Synthetic Audio book
(iii) PDF compression
(iv) Audio Merger

The web app is hosted on a 2019 microsoft ~~apache web server~~ cloud server. The apache 24 web server frame work is used to integrate the web app with the domain / IP address the cloud server is associated; allowing web traffic to flow via Port 81 & Port 413 or any other customized port as long as its configured in the

Once the apache framework integrated the cloud server to the django web app, django web app became production deployable.

Web-app

↳ The web-app is built of a couple components;

(i) Front-End GUI

(ii) Rest API's integrating to the Pythonic Backends.

(iii) Encryption system, which allows production scalability for infinite users as well as a data governance system.

(iv) Django middleware which facilitat Rest API and pythonic / C#/C backe. integration.

(v) Redis database to run each request asynchronously.

Front End
GUI

Allows access to 4 main Rest API's connected to their own backends built using HTML, CSS, Javascript, etc.

Front-end GUI's include loading pages, users can interact with while task is processing, as well as download pages when JSON outputs finish processing.

—

Rest API's

↳ Connects the user information, ~~Sucho~~ such as file uploads to the pythonic API's / modules to produce Audio book or requested output based on input.

Rest API 1:

1) Conventional Audiobook:

2 choices: conventional or custom

(i) Conventional:

This takes a file per user specification & uploads to a web-server directory.

=> This file is then given to the integrated "tesseract OCR" engine.

=> Uses Computer Vision and Image Recongntion to map latin characters to python string, alphanum character producing a string.

=> This string is saved as an object variable, and passed on to the pyttx API, which produce the .wav files, which are then merged to form your audiobook.

- Audiobook saved as an .mp3, par saved in the directory under specific user instance thanks to the encryption system, which is then passed back to the REST API as a JSON response, as downloadable file.

- Custom is the same process, except user specifications of pyttx API configurations, such as voice sex, volume, sound, ~~etc~~ & output format are passed from the Rest API to the pyttx API configurations as python objects via django middleware and built in ORM (object relational management) database. This allows the user to produce a somewhat customizable Audiobook.

Rest API 2

) Voice Synthesis:

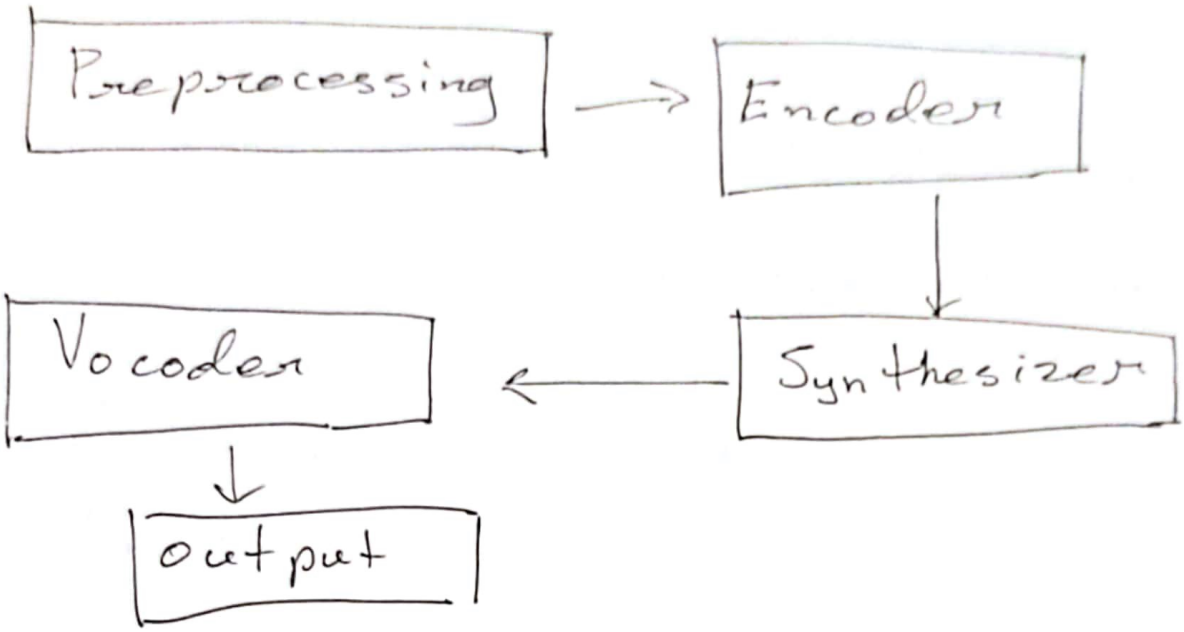
Rest API uses django middleware to take user file submission of training data & pdf to make an audio file, & saves the files under user instance.

↳ The pdf is feed through the "tesseract "OCR" engine to produce a python object with the text data, passed through the django framework via ORM database to the custom synthesizer.

~~Custom~~

P.T.O

Custom Synthesizer works in the following way:



The Preprocessing phases basically produceses the python string for the audio clip.

The encoder takes phonetic and Latin linguistic ~~mopre~~ trained data to produce a phonetic model, so basically it uses a neural network to produce a syllabul model, ~~Latin characters : sounds~~

Syllabel model:

data then taken from syllabel model and thrown into another LTSM neural network to produce a linguisitic model. (Linguistic model is essentially a giant dictionary, with Latin words as keys, and audio as values, once a key is accessed, the audio signals mess together to produce a .wav file.

Synthesizer:

Takes syllabel model transforms it into a linguistic model to be used to produce speech.

vocoder : Takes provided pythonic string object & runs it to get a synthetic voice.

Now, takes audio_data provided by the user instance in the Rest API, breaks down the phonetic data, adds it the synthetic model and shifts the weight, of the phonetic frequencies stored in the mel spectrogram, and adjust the synthesizer based on training data.

Python object is thrown into the new custom synthesizer, producing audio data similar to the audio training file.

↳ Creating an audiobook with a cloned voice.

-> This new cloned voice file is then pas saved in the internal output directory under the user instance, which is then passed back through the django middleware to the Rest API, which passes it to the ~~#~~ user instance of a Front-End GUI as a JSON output, i.e ~~d~~ a downloable audio file.

Rest API 3

PDF compression:

↳ Takes PDF submission from Rest API request generated by the user and saves pdf(s) under user instance before turning file system location data into python objects.

↳ python object then interfaced with a ghostscript engine via python sub-module.

↳ ghostscript engine written in typescript, designed to reduce PDF size without loss of quality by identifying unnessecary data and eliminating it.

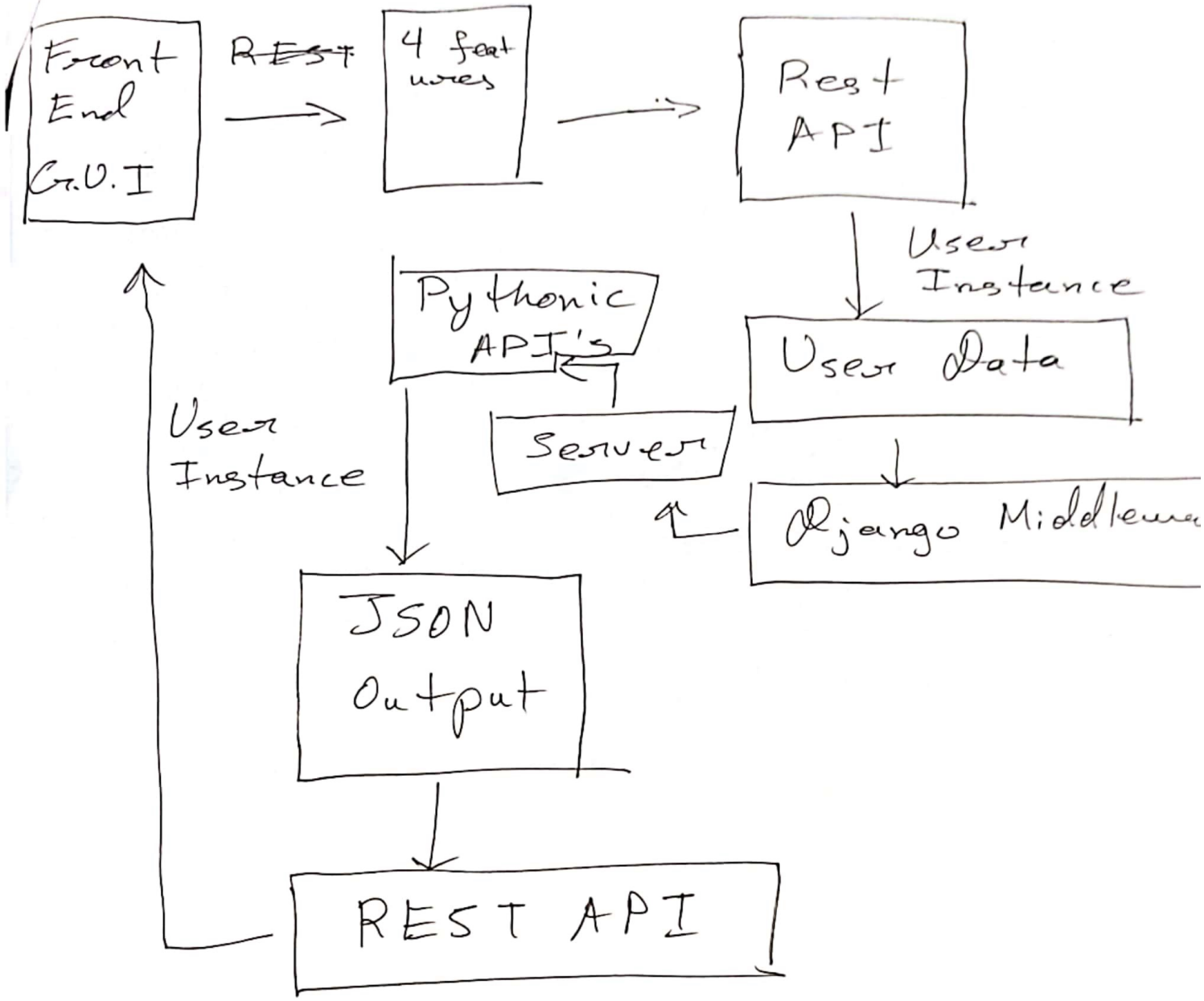
↳ Post compression, PDF's saved into a compressed folder.

- Compressed Folders sent back to user instance via django middleware as JSON output as a downloadable compressed zip file.

Rest API 4

=> Audio clips passed from Rest API through django API middleware to CLI & moviepy; audio clips are decoded and merged together into 1 larger audio clip in sequence, this larger audio file is then thrown back through the django middleware from the saved user instance and served to the user as a flow .JSON output as a download file.

Schematic


Front End G.U.I
REST
4 features
Rest API
User Instance
User Data
Pythonic API's
Server
Django Middleware
User Instance
JSON Output
REST API