Dokumentation Taste Waves

Backend Dokumentation

Setup

Um das Backend aufzusetzen musste zuerst Python installiert werden:
https://www.python.org/downloads/
Danach kann PIP installiert werden, was uns die Installation von Paketen um einiges erleichtert:

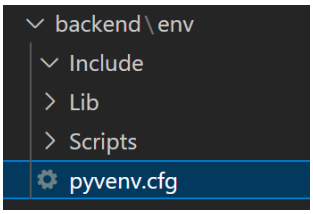
MAC/OSX: `python3 -m pip install`
Windows: `py -m pip install`

Dann wird das Package für das Virtual Environment installiert:

Windows:`pip install virtualenv`
MAC/OSX:`sudo python2 -m pip install virtualenv`

Jetzt kann der Ordner erstellt werden, indem sich das Backend befindet.
Das Terminal wird geöffnet und in den Backend Ordner navigiert, hier wird der Befehl zur erstellung der virtuellen Umgebung eingegeben: python -m venv env.

Um die virtuelle Umgebung zu aktivieren, muss die “activate.bat” Datei ausgeführt werden. Dazu wird folgendes in das Terminal eingegeben: "Scripts/activate.bat". Die Ordnerstruktur sollte nun so aussehen:



Über PIP können wir Flask installieren: “pip install Flask”

Um zu testen ob alles funktioniert hat kann nun eine Python Datei angelegt werden:

Dieser Code bildet die Grundstruktur einer Flask App:

```python
from flask import Flask, flash, request, json, jsonify, send_file,
send_from_directory  #Basis API Importe

app = Flask(__name__)

if __name__ == '__main__':
    app.run(debug=True)
```

Mit folgendem Befehl lässt sich die Flask App starten: “python main.py”.
In der Konsole sollte nun das zu sehen sein:

```
PS C:\Git\test-folder\backend\env> python main.py
 * Serving Flask app 'main'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 236-377-933
 * Detected change in 'C:\\Git\\test-folder\\backend\\env\\main.py', reloading
 * Restarting with stat
```

Backend Implementierung

Damit alles reibungslos funktioniert, werden folgende Packages importiert, welche vorher über pip install <Paketname> installiert werden.

```python
from flask import Flask, flash, request, json, jsonify,
from werkzeug.utils import secure_filename  #Import um
from flask_cors import CORS #Inportiert CORS um Cross-O
from moviepy.editor import * #Import um das Audio aus d
from pydub import AudioSegment
import math
import os #Um das File welches eintrift zu speichern
```

Damit unsere App auf Post Requests reagiert brachen wir ein Pfad welcher angesprochen werden kann, dazu benutzen wir eine Annotation:

```python
#Upload Route um das File auch hochzulade
@app.route('/upload', methods=['POST'])
```

Einerseits werden der Pfad “/upload” und die Methode, auf welche unsere App reagiert, definiert: “Post”. Nun kann definiert werden, was bei einem Post-Request auf diesem Pfad passiert. Hierzu wird eine Methode erstellt, welche gleich heisst wie der Pfad. Die Methode beinhaltet die Logik, um ein Video entgegenzunehmen und das Audio von dem Video zu trennen.

```python
file = request.files['file']#Das file wird entgegen genommen

if file: #Falls das File gesetzt ist wird es abgelegt

    filename = secure_filename(file.filename) #Der Name wird sicher abgespeichert

    file.save(os.path.join('static/uploads', filename)) #Das File wird gespeichert

    vid = VideoFileClip('static/uploads/'+filename) #Das File wird nochmals geholt
    audio = vid.audio

    defName = filename.split('.')[0]
    audio.write_audiofile('mat/audio/'+defName+'.wav') #Das Audio File wird erstellt
```

Um das Audio in eine Waveform umzuwandeln, werden die Informationen dieser Datei benötigt. Da Files encoded werden, um sie wieder an das Frontend zu schicken und dies nicht ideal ist, muss die Information hier extrahiert werden. Dazu wird nur ein Kanal benötigt, da die Menge an Daten ansonsten zu gross wäre.

```python
audio_file = AudioSegment.from_wav(open("mat/audio/"+defName+".wav", "rb"))
audio_file = audio_file.split_to_mono() #Es wird nur ein Kanal benoetigt des
data = audio_file[0]._data
```

Als nächstes werden die encodierten Werte dekodiert und in ein Array gespeichert.

```python
#Die Daten welche wir aus dem Audio File erhalten haben werden nun in Integers verwandelt
for sample_index in range(len(data) // 2):
    sample = int.from_bytes(data[sample_index * 2: sample_index * 2 + 2], 'little', signed=True)
    data_values.append(sample)
```

Jetzt wird die Menge an Daten reduziert und in verarbeitbare Werte umgewandelt.

```python
sample_size = 70

#Aus der gesamt Laenge des Integer Array wird benoetigt um die Block laenge
block_size = math.floor(len(data_values) / sample_size)

half_filtered_data = []
filtered_data = []

#Die Block laenge wird genutzt um einen kleineren Datensatzt zu erhalten
for i in range(sample_size):
    half_filtered_data.append(data_values[i * block_size])

#Die Daten werden nun verkleinert
for j in range(sample_size):
    filtered_data.append(abs(math.ceil(half_filtered_data[j] / 10)))
```

(variable) block_size: Any

Die Daten sind nun bereit gezeichnet zu werden, dazu senden wir die Daten an das Frontend:

```python
#Die Daten werden nun zurueck an den Client gesendet
return filtered_data
```

Frontend Dokumentation

Setup Frontend

Das Frontend benötigt Node.js: https://nodejs.org/en/
Über den Befehl: “npx create-react-app my-app” kann die React App aufgesetzt werden. Alle out of the box Dateien werden gelöscht, zudem werden alle Imports auf die gelöschten Dateien entfernt. Benötigt wird Axios, dies erleichtert die Requests auf das Backend. Hierzu wird der Befehl “npm install Axios” genutzt. Die Dateiformate, welche im Frontend zum Upload erlaubt sind, sind ausschliesslich “mp4” Dateien, sie sind das gängigste Videoformat, deshalb wird es hier benutzt.

Frontend Implementierung

Um die Request durchführen zu können ist es nötig Axios zu konfigurieren:
Dazu wird ein File mit dem Namen axios.js angelegt, darin definiert ist die base URL.

```
import Axios from "axios";



export const axios = Axios.create({
    baseURL: "http://127.0.0.1:5000"
});
```

Die Struktur der Seite beinhaltet folgendes: einen Titel mit zwei Bildern, darunter ein Input-Feld getarnt als Button zum Hochladen der Dateien, ein Button zum senden und ein ungefüllter Canvas. Das HTML Gerüst befindet sich in einer Funktion namens Render, diese baut die Struktur der Seite im Web auf. Die Tags sind dieselben wie in einem normalen HTML-File, jedoch befindet sich der HTML-Code mit der React Library in einem jsx File. Die Buttons sowie das Input-Feld besitzen Events, welche jeweil die Methoden handleInput() und handleUpload() aufrufen.

```jsx
<div className="upload">
  <div className="title">
    <img src={wave} alt="" className="img2" />
    <h2>Taste Waves</h2>
    <img src={wave} alt="" className="img1" />
  </div>
  <div className="text">
    <p>Here you can taste your favorite video's audio, just upload your video here and press send</p>
  </div>
  <form>
    <div className="form">
      <label htmlFor="file-upload" className="custom-file-upload">
        <i className="fa fa-cloud-upload"></i> Upload
      </label>
      <input type="file" name="file" id="file-upload" onChange={this.handleFile}></input>
      <button type="button" onClick={this.handleUpload} className="send">Send File</button>
    </div>
  </form>
  <canvas id="canvas" className="canvas"></canvas>
</div>
```

Das Design ist schlicht, der Hintergrund ist dunkel und die Schrift sowie das Bild der Waveform weiss. Die Abstände sind definiert und ein visueller effekt ist hinzugefügt um das drücken der Buttons hervor zu heben, ausserdem werden die Schriftgrössen und die Waveform angepasst wenn der Screen kleiner ist:

```css
.send:hover {
    box-shadow: 0 0 11px rgba(240, 235, 235, 0.2);
}

.form {
    display: grid;
    justify-items: center;
}

@media only screen and (max-width: 600px) {
  h2 {
    font-size: 60px;
  }

  .text {
    display: grid;
    justify-items: center;
    font-size: 20px;
  }
}
```

Das Video wird wie folgt verschickt, zuerst wird der Button zum auswählen des Videos gedrückt, welcher den State, der zur Verwaltung der Daten zwischen den Komponenten dient, auf das Video gesetzt wird. Zusätzlich wird der Name gesetzt, falls dieser Null ist erscheint kein Send Button, zusätzlich wird überprüft, ob der Name die File-Extension “.mp4“ hat.

```
//Hier wird der State auf das File gesetzt
handleFile = (e) => {
  let file = e.target.files[0];
  let filename = file.name;

  if(!filename.endsWith('.mp4')) {
    alert('You can upload video files only.');
    return;
  }

  console.log(file);
  this.setState({ file: file, filename: filename});
}
```

Der nächste Button schickt das File über Axios zum Backend, dies geschieht asynchron. Das File wird aus dem State einer Variable übergeben und einem Form-Data-Objekt angehängt. Einer Konstante wird dann der Wert aus der Response der Request übergeben, über welche die Methode createCanvas() dann die Waveform zeichnet.:

```
handleUpload = async () => {
  let file = this.state.file;
  let formData = new FormData();

  formData.append('file', file, file.name);

  const res = await axios.post('/upload', formData).catch((err) => {
    console.log(err);
  });

  console.log(res.data);
  //Wenn die Response erhalten wurde wird die Kurve gezeichnet
  this.createCanvas(res.data);
}
```

Wenn die Werte des Audio Files im Frontend eintreffen, wird die Zeichnen-Methode aufgerufen, diese benötigt ein Canvas, welches so angepasst ist, dass das Zeichnen in der Mitte des Canvas geschieht. Der Canvas ist so konfiguriert, dass dieser sich dem Gerät anpasst. Bei einer geraden Zahl wird die Kurve nach oben verlaufen, ansonsten nach unten.

Die Höhe entspricht den aus dem Backend kommenden Zahlen.

```
const dpr = window.devicePixelRatio || 1;
const padding = 20;
canvas.width = canvas.offsetWidth * dpr;
canvas.height = (canvas.offsetHeight + padding * 2) * dpr;
//Hoehe und Breite wird nach dem Bildschirm Mas gezeichnet
const ctx = canvas.getContext("2d");
ctx.scale(dpr, dpr);
//Hiermit wird die Mitte des Canvas auf das optische Zentrum g
ctx.translate(0, canvas.offsetHeight / 2 + padding);
const width = canvas.offsetWidth / data.length;

for (let i = 0; i < data.length; i++) {
  const x = width * i;

  let height = data[i];
  if (height < 0) {
    height = 0;
  } else if (height > canvas.offsetHeight / 2) {
    height = height > canvas.offsetHeight / 2;
  }
  //Die Kurve wird hier dann richtig gezeichnet
  this.drawLineSegment(ctx, x, height, width, (i + 1) % 2);
}
```

```
drawLineSegment = (ctx, x, y, width, isEven) => {
  ctx.lineWidth = 2; //Linien dicke
  ctx.strokeStyle = "#fff"; // Farbe der Kurve
  ctx.beginPath();
  y = isEven ? y : -y; //Wenn die Zahl gerade ist geht
  ctx.moveTo(x, 0);
  ctx.lineTo(x, y);
  ctx.arc(x + width / 2, y, width / 2, Math.PI, 0, isEv
  ctx.lineTo(x + width, 0);
  ctx.stroke();
};
```

Mit diesen Zeilen wird das Canvas nun heruntergeladen, zuerst wird ein Link erstellt, danach wird der Name gesetzt. Dem Link wird das Canvas zugewiesen, sodass das Bild per Link heruntergeladen werden kann. Der Link wird schlussendlich automatisch geklickt:

```
var link = document.createElement('a');
link.download = 'filename.png';
link.href = document.getElementById('canvas').toDataURL()
link.click();
```

Das Resultat sieht so aus:

Reflexion

Gesamthaft verlief das Projekt gut, ich habe viele neue Dinge gelernt und meine gestellten Ziele fast vollständig erreicht, lediglich der Download mit der URL ist noch zu erfüllen. Mein Wissen wurde hier ebenfalls sehr vertieft, gerade was den File Transfer von Videos angeht. Schwierigkeiten gab es auch, deshalb kann man sagen, habe ich hier auch erfahren, wie ich mich verhalten muss, wenn ich nicht weiter komme.

Gerade der File Transfer war eine Herausforderung, probiert habe ich es mit Multer und Nestjs, jedoch waren die Dateien zuerst zu gross, weshalb ich Multer als Library hinzu zog. Nun gab es ein Problem mit dem Speichern des Files, was mit in 7-bit encoding geschah, jedoch fand ich keinen Weg, das gespeicherte File in ein brauchbares Format zu konvertieren.

Ich entschied mich dann für einen Wechsel. Ich nahm mir vor die API in Flask, ein Python Framework umzusetzen, was Files in dieser grösse auch in mp4 abspeichern konnte. Dadurch erweiterte ich mein Wissen in Python.

React bot mir weniger Probleme, die leicht lernbare Struktur erleichterte mir das Umsetzen.

Mein Zeitmanagement lässt noch zu wünschen übrig, jedoch konnte ich meine Effizienz steigern, was sich in der letzten Phase der Projektumsetzung als grossen Vorteil herausstellte. Ich blib länger und hartnäckiger dran, was ich als grosses Plus aus dem Modul mitnehme.