Technische Dokumentation

Das folgende Dokument beschreibt die Modulrelevanten Grundfunktionen meiner Projektarbeit. Einige als wichtig erachtete Punkte werden ausführlich beschrieben, die Dokumentation ist aber alles andere als vollständig. Im besten Fall bietet sie einen schnellen Einstieg in den Code.

1. Grundstruktur

Das Projekt besteht aus einer rein clientseitigen App, sämtlicher Code ist in Javascript geschrieben. Grundsätzlich finden sich folgende von mir geschriebene Komponenten:

index.html	Lädt sämtliche Javascripts und definiert einige spezielle Stile. Insbesondere wird hier für hochauflösende Bildschirmtypen die Schriftgrösse des UI erhöht.
app.js	Einstiegspunkt der App. Stellt sicher, dass der Browser des Clients die nötigen APIs unterstützt (Audio API und WebGL). Enthält den Rendering-Loop und setzt das UI auf.
webgl.js	Wrapper um three.js. Initialisiert eine Szene, eine Kamera und einen Renderer und enthält Funktionen zum Malen verschiedener Elemente wie Linien, Rechtecke oder Punkte.
audio.js	Wrapper um die Audio-API. Behandelt das Hochladen und die Analyse von Audiodateien und stellt Funktionen für die Audiovisualisierung zur Verfügung, die dann wiederum auf die Funktionen in webgl.js zurückgreifen.
animatedCircle.js	Dieses Script steht zwischen audio.js und webgl.js: Es beinhaltet Funktionen zum Malen eines von Audio animierten Kreises und einer Spirale. Dieser Code wurde relativ gross und wurde deshalb aus audio.js augelagert.

2. Dateiformate und Upload

Audiodateien werden vom Benutzer via Drag and Drop hochgeladen. Dem Drop-Event wird im Konstruktur von audio.js ein Eventlistener hinzugefügt:

```
window.addEventListener('drop', (event) => {
        ...
}
```

Wichtig ist nun in den folgenden Codezeilen das Objekt «event.dataTransfer». Hier prüfe ich zunächst, ob überhaupt ein DataTransfer vorhanden ist und ob es sich dabei um eine Datei handelt. Bei einem DataTransfer können auch mehrere Files vorhanden sein – dies ignoriert die App, sie wählt schlicht die erste vorhandene Datei.

In einem zweiten Schritt wird der Dateityp geprüft. Eine Liste der zugelassenen Dateitypen wurde bereits im Konstruktor der Audio-Klasse definiert:

```
this.allowedFileTypes=['audio/mpeg', 'audio/mp3', 'audio/wav', 'audio/mp4', 'audio/ogg' ]
```

Versucht der Benutzer, einen anderen Filetyp zu laden, wird dies mit einer Fehlermeldung quittiert. Bewusst nicht geprüft wird hingegen die Dateigrösse: Da die Website nur clientseitig arbeitet, werden ja keine Daten im eigentlichen Sinn hochgeladen und wir müssen keinen Platz sparen. Auch eine tiefere Prüfung des Dateityps, die über den MIME-Type hinweg geht (zum Beispiel öffnen der Datei und prüfen des Headers), ist hier nicht nötig.

Nach dieser Prüfung werden die Dateidaten an die Funktion «loadAudio(data)» weitergegeben.

3. Audioanalyse

3.1 Vorbereitende Schritte (Funktion loadAudio(data))

Um unsere Audiodaten zu analysieren, brauchen wir drei Vorbereitungsschritte. Zunächst erzeugen wir im DOM ein Audioelement und weisen ihm als Source die vom Benutzer gedroppte Datei zu:

```
this.htmlAudio = document.createElement('audio')
// ...
this.htmlAudio.src= URL.createObjectURL(data)
```

Danach erzeugen wir mittels Audio-API einen Audio-Kontext. Dem Kontext müssen wir wiederum das html-Element zuweisen. Erst jetzt können wir einen Audio-Analyser generieren und ihn mit unserer Audiosource verbinden. Dieser Vorgang ist im Code gut dokumentiert (audio.js, Zeilen 92 bis 109).

3.2 Analyse

Die Audioanalyse geschieht mit Hilfe der Web-Audio-API, die von den meisten Browsern (ausser InternetExplorer) unterstützt wird[1]. Für das Projekt benötigte ich nur sehr wenige Funktionen dieser API.

Die Audio-API bietet zwei Arten von Analysen an: Frequenzdaten und Wellenform. Die Frequenzdaten werden in audio.js in der Funktion «getFrequencyData() geholt. Sie sind etwas schwer zu interpretieren[2]. Grundsätzlich wird hier ein Array erstellt, dessen grösse wir über «analyser.fftSize» bestimmen können. Die Daten im ersten Arrayindex repräsentieren die Lautstärke bei der tiefsten Frequenz, die Daten im letzten Arrayindex die Lautstärke in der höchsten Frequenz, wobei hier jeder Datenpunkt einen ganzen Frequenzbereich abdeckt, dessen Grösse sich auf der «fftSize» ergibt - Dies habe ich leider erst spät verstanden, sonst hätte ich mit diesen Daten mehr anfangen können.

Die Wellenform muss nicht weiter erklärt werden. Sie wird in der Funktion «analyseFrequencyData» geholt. Auch hier liefert uns die Audio-API ein Array aus Datenpunkten für einen jeweiligen Zeitschritt.

1 https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API
2 Vergleiche https://stackoverflow.com/questions/44502536/determining-frequencies-in-js-audiocontext-analysernode

6. Audiosvisualisierung

6.1 Initialisierung

Im vorigen Kapitel haben wir grob gesehen, wie der Audioinput untersucht wird. Nun geht es darum, die daraus gewonnen Daten zu visualisieren. Dazu verwende ich die three.js-Bibliothek, die wiederum auf WebGL aufbaut[3]. Initialisierung und Basisfunktionen von three.js finden sich «webgl.js».

Im Konstruktor der Klasse WebGL ist zunächst wichtig, dass sowohl die Kamera als auch der Renderer die Grösse des Fensters kennen, in das sie malen. Dies wird auch beim Thema Responsivität wichtig. Die Kamera muss das Verhältnis von Breite und Höhe des Malfensters kennen, um die Proportionen von geometrischen Figuren zu erhalten:

```
this.camera=new THREE.PerspectiveCamera(45,
                window.innerWidth/window.innerHeight,
                0.1,100)
```

(Der erste Parameter bestimmt den Winkel, den die Kamera abdeckt: Mit einem engeren Winkel liesse sich in die Szene zoomen. Die letzten beiden Parameter bestimmen, ab welcher und bis zu welcher Tiefe die Kamera ein Objekt malt, was in unserem Fall nicht wirklich relevant ist, da wir in 2D bleiben).

6.2 Rendering

Alle Objekte, die mit three.js gemalt werden, bestehen aus zwei Eigenschaften: Einem Material und einer Geometrie. Wurde ein Objekt definiert, muss es an die Szene von three.js angehängt werden, um gemalt zu werden. Ich verwende für die meisten meiner Formen simple Materialen. Für das Plotten der Wellenform etwa benutzte ich in webgl.js die Funktion «drawLineStrip(colors, vertices)» mit dem Material:

```
var material = new THREE.LineBasicMaterial({
        color: color
});
```

Der animierte Kreis oder die Spirale hingegen werden mit THREE.PointsMaterial erstellt. Hier wird auch eine einfache Textur mit Alphakanal geladen, um einen Blendeffekt zu erzeugen.

Hier ein kurzer Überblick über die Visualierungsmöglichkeiten der App und die entsprechenden Funktionen in audio.js und webgl.js:

Visualisierung	audio.js	webgl.js
Audio Wave	drawWaveform()	drawLineStrip(color, vectors)
Frequency Bar	analyseFrequencyData()	drawBars(color, vectors)
Animated Circle	drawAnimatedCircleFromWave()	Klasse «Particle»
Animated Spiral	DrawAnimatedSpiralFromWave()	Klasse «Particle»

3 https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene

Kreis und Spirale greifen zudem auf die Hilfsklassen AnimatedCircle und AnimatedSpiral im Script «animatedCircle.js» (ein sehr überarbeitungsbedürftiges Script). Der Parameter «vectors» in der Tabelle oben stellt die Geometrie meiner Objekte dar und besteht stets aus einem Array aus «THREE.Vector3»-Objekten, deren x- und y-Koordinaten in «audio.js» berechnet werden.

7. User Interface

Das User Interface ist recht schlicht gehalten und beinhaltet zwei Elemente: Links oben finden sich Checkboxen und Sliders, mit der sich die Audiovisualisierung beeinflussen lässt. Dieses «Menü» wird mit Hilfe der Bibliothek dat.gui[4] im Script «app.js» im Konstruktor der Klasse «App» erstellt.

Zum zweiten wird der Benutzer manchmal mit Statustexten der App informiert. Beispiel: Eine neue Audiodatei wurde erfolgreich geladen. Diese Texte werden ebenfalls mit einer externen Bibliothek, nämlich mit «notify.js» erstellt[5].

8. Responsivität

Die Audioausgabe an sich besteht nicht aus einem Bild, sondern aus Tönen, und entzieht sich deshalb dem Thema Responsivität. Für die Elemente «Rendering mit three.js» und für das User Interface hingegen bestehen zwei grundsätzlich verschiedene Ansätze.

8.1 Grafik mit three.js

Sämtliche Grafikobjekte werden vom THREE.Renderer gemalt, der dazu wiederum eine THREE.Kamera braucht. Beide müssen, wie in Kapitel 6 bereits beschrieben, an den Bildschirm angepasst werden. Damit wird auf jedem Bildschirm eine fast identische Szene gemalt – ein vollständig identische Szene ist aber nicht möglich, ohne die Proportionen der Geometrien zu verzerren (ein Quadrat soll ja sowohl auf Breit- als auch auf Hochbildschirm ein Quadrat bleiben).

Gerade im Hochformat auf einem Handy werden die gerenderten Szenen teilweise am Bildschirmrand abgeschnitten. Hier informiere ich den Benutzer mit einem Statustext, dass die App im Breitformat besser aussehen wird. Für die Kreis- und Spiralanimationen kann der Benutzer zudem den Kreisradius im User Interface bestimmen und damit die vollständige Szene auf jeden Bildschirm anpassen.

Zu erwähnen ist hier noch die Funktion «resize» in Script webGL.js, die wiederum in Script «app.js» als Eventhandler eines Fenster-Resizes registriert wird: Sie passt den Viewport der Kamera und des Renderers wieder an.

8.2 User Interface

Leider hat sich die verwendete Bibliothek dat.gui als recht unresponsiv erwiesen: Es wird stets die selbe Schriftgrösse verwendet, was zum Beispiel auf Retina-Displays zu einer sehr kleinen Schrift führt. Das Projekt gleicht diesen Mangel teilweise aus, indem das CSS von dat.gui überschrieben wird. Der Stil ist unschönerweise in «index.html» definiert und erhöht die Schriftgrösse je nach Pixel-Ratio des Displays.

4 https://github.com/dataarts/dat.gui
5 https://notifyjs.jpillora.com/