Fiche technique SpeechToPicto

Cette fiche a pour objectif de présenter les différents fichiers ainsi que les fonctionnalités qui y sont implémentées.

On y retrouve plusieurs composants principaux :

- speech-to-picto
- audio-text-file

Ainsi que des services :

- audioRecorder
- audioTextFileShare

On retrouve aussi dans les assets le répertoire Whisper que j'expliquerais à la fin.

La page pour ces fonctionnalité a comme URL /speechToPicto (on peut le voir dans le fichier app-routing.module.ts à la racine du répertoire app.

Service audioRecorder

le but de ce service est d'implémenter MediaRecorder et SpeechSynthesisUtterance, ses attributs les plus importants sont :

```
- mediaRecorder!: MediaRecorder;
- audioChunks: any = [];
- audioBlob: Blob = new Blob();
- audioUrl: any;
- audio = new Audio();
```

- MediaRecorder : Instance de MediaRecorder utilisée pour gérer l'enregistrement de l'audio.
- audioChunks : Tableau pour stocker les morceaux d'audio enregistrés.
- audioBlob : Blob contenant les données audio enregistrées.
- audioUrl : Url pour accéder au blob.
- audio : Instance d'audio, pour lire l'audio enregistré.

Contient les fonctions les plus importantes :

```
startRecording() {
 this.audioChunks = [];
 navigator.mediaDevices.getUserMedia({audio: true})
  .then(stream => {
    this.mediaRecorder = new MediaRecorder(stream);
    this.mediaRecorder.start();

    this.mediaRecorder.addEventListener("dataavailable", event => {
      this.audioChunks.push(event.data);
    });

    this.mediaRecorder.addEventListener("stop", () => {
      this.audioBlob = new Blob(this.audioChunks, {type: "audio/wav"});
      this.audioUrl = URL.createObjectURL(this.audioBlob);
      this.audio = new Audio(this.audioUrl);
    });

  });
}
```

startRecording() permet de démarrer l'enregistrement audio.

Il fonctionne selon les étapes suivantes :

- Réinitialise audioChunks pour stocker les nouveaux morceaux d'audio.
- Utilise navigator.mediaDevices.getUserMedia pour demander l'accès au microphone.
- Initialise MediaRecorder avec le flux audio obtenu.
- Commence l'enregistrement avec mediaRecorder.start().
- Ajoute un écouteur d'événements dataavailable pour collecter les morceaux d'audio.
- Ajoute un écouteur d'événements stop pour créer un Blob avec les morceaux d'audio et générer une URL utilisable pour lire l'audio.

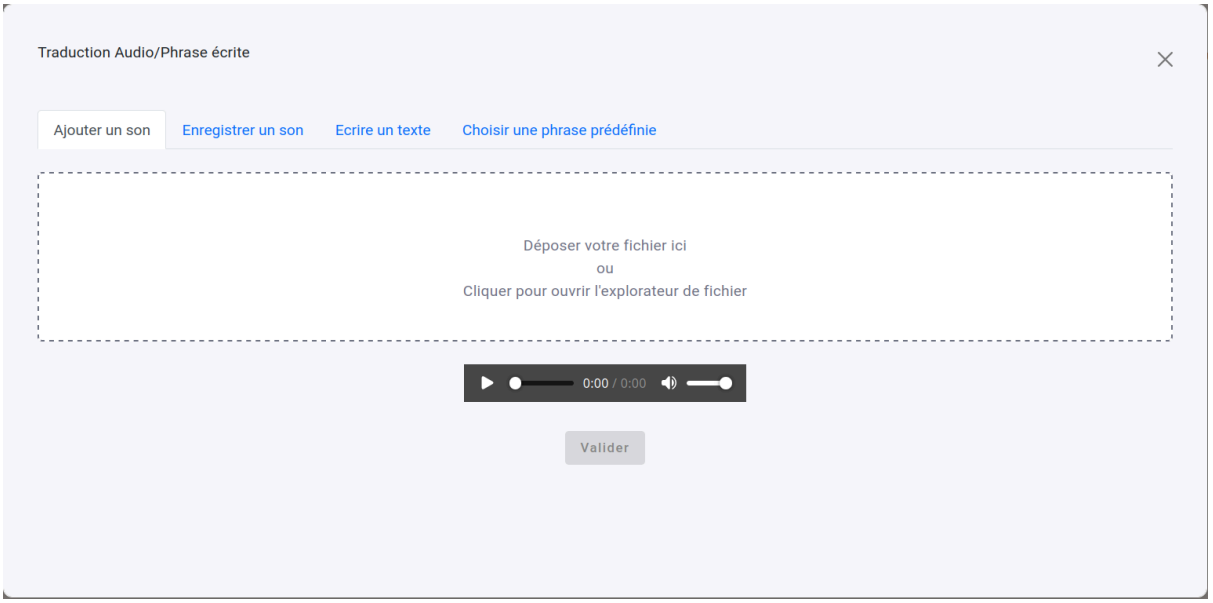
Les autres fonctions sont moins importantes :

- stopRecording() = arrête l'enregistrement.
- listenRecording() = lis l'audio (non utilisé pour le moment)
- speechSynthesis() = Convertit lee texte en parole.

Composant speech-to-picto

Ce composant a pour objectif de contenir le front-end principal, c'est aussi ici que doivent les pictogrammes être affichées.

Il contient un bouton qui permet d'appeller le code présent dans un autre composant (audio-text-file), qui affiche le menu pour la traduction écrite / audio.



Voici un screenshot du susmentionné menu.

Composant audio-text-file

Le screenshot du menu ci-dessus est tiré du code du composant audio-text-file.

La lemmatisation est intégré dans ce composant, de plus, ce composant permet la gestion des fichiers audio via enregistrement ou encore la sélection d'un fichier audio.

```
dropFile(value: any) {
 const song = value.addedFiles[0];
 const reader = new FileReader();
 if (this.checkFile(song)) {
  this.showErrorDropFile = false;
  try {
   reader.readAsDataURL(song);
   reader.onload = () => {
    this.showFileUpload = true;

    let progressUpload: number = 0;
    const progressInterval = setInterval(() => {
     if (progressUpload < 99) {
      progressUpload += 5;
      this.uploadFileProgress = "width: " + progressUpload + "%";
     } else {
      clearInterval(progressInterval);
      setTimeout(() => {
       this.nameSound = song.name;
       this.soundToListen = String(reader.result);
       //this.soundToZip = song;
```

```
          this.showFileUpload = false;
          this.showFile = true
          this.disableAddSongButton = false;
        }, 1000);
      }
    }, 200);
  };
 } catch (e) {
  this.errorDropFile = " Le fichier est corrompu, impossible de le charger ! ";
  this.showErrorDropFile = true;
 }
} else {
 this.errorDropFile = " Le fichier n'est pas un format que l'on accepte, mp3 ou wav seulement ! ";
 this.showErrorDropFile = true;
}
}
```

Cette fonction permet de gérer le dépôt de fichier.

Plus précisément, elle :

- Vérifie le type de fichier audio.
- Lit le fichier et affiche une barre de progression pour le téléchargement.
- Met à jour les propriétés nameSound et soundToListen une fois le téléchargement terminé.

```
recording() {
 if (this.isRecording) {
  this.stopRecord();
 } else {
  this.startRecord();
 }
}
```

Cette fonction permet d'appeler les fonctions permettant d'arrêter ou de démarrer l'enregistrement audio.

Service audioTextFileShare

Ce service, de par l'utilisation d'observables afin de transmettre les valeurs des variables du composant audio-text-file vers le composant speech-to-picto.

En l'état c'est utilisé pour transmettre la valeur de la lemmatisation, mais cela sera aussi utile pour transmettre le fichier audio et peut-être le Tuple de sortie des scripts de TAP.

Scripts de Traitement Automatique de la Parole (TAP)

Ces scripts permettent le traitement des fichiers audio.

Ils se situent dans le répertoire src/assets/whisper. Ils permettent de traîter le fichier audio afin d'en sortir un Tuple qui sera utilisé pour la communication avec l'API Arasaac dans le cadre de l'affichage des pictogrammes.

Les scripts sont à exécuter dans l'ordre suivant :

script whisper -> run_TA -> mapping

Concernant le run_TA, étant un fichier très volumineux, il n'est pas push dans le Github, mais doit être implémenté sur le serveur directement.

Exemple de sortie de tuple final :

[(7095, 'ceci'), (8114, 'est'), (2627, 'un'), (0, 'test')]

commande utilisée (en étant placé dans le bon répertoire):

python3 mapping_pictograms.py --text "ceci est un test" --lexicon "lexique.csv"

(pour pouvoir voir ceci via la console, il faut décommenter les print présent dans les scripts.)