**بسم الله الرحمن الرحیم**

موضوع: گزارش پروژه پیاده سازی تعامل صوتی با ChatGPT

استاد راهنما : سرکار خانم بهشید بهکمال

توسعه دهنده : رضا نصیری

فهرست

مقدمه

تکنولوژی های پروژه

- برنامه نویسی وب

- برنامه نویسی سرور

- سوکت

- ChatGPT

- تبدیل متن به صدا

- تبدیل صدا به متن

**مقدمه**

هوش مصنوعی این روزها به سراسر زندگی بشر نفوذ کرده است ؛ خودروهای خودران ، چت بات‌ها ، دستیارهای صوتی و ... نمونه

ای از این موارد می باشند چت‌ بات‌های هوش مصنوعی یکی از این موارد هستند که در آن انسان با یک سیستم مبتنی بر هوش

مصنوعی ارتباط برقرار می کند و سیستم به قدری هوشمندانه به سوالات کاربر پاسخ می دهد که کاربر به سختی می تواند

تشخیص دهد که طرف مقابل خود یک ربات می باشد.

تکنولوژی ChatGPT جدیدترین محصول در حوزه هوش مصنوعی می باشد که در دنیا فراگیر شده است و مردم را حیرت زده

کرده است . این ربات می تواند به تمام سوالات کاربران در تمام حوزه‌های مختلف مثل برنامه نویسی ، ریاضی ، تاریخی ، اطلاعات

عمومی و ... پاسخی جامع و کامل ارائه دهد.

این ربات که توسط شرکت OpenAI ساخته شده است در اخرین نسخه‌ی خود مبتنی بر مدل زبانی GPT 3.5 می باشد. این

مدل زبانی که بر پایه‌ی معماری شبکه های عصبی ساخته شده است قادر به پردازش زبان طبیعی و تولید متن می باشد.

اگر با ChatGPT کار کرده باشید خواهید فهمید که این بات مبتنی بر تعامل متنی می باشد و شاید شما علاقه ای به پرسیدن

سوالات خود به صورت متنی نداشته باشید و بخواهید که به صورت صوتی سوالات خود را بپرسید. سیستم پیاده سازی ای در این

پروژه قصد گزارش آن را داریم این امکان را به شما می دهد که سوالات خود را به صورت صوتی از ChatGPT بپرسید.

**ساختار پروژه**

این برنامه یک سیستم نرم افزاری تحت وب می باشد که بر پایه‌ی معماری client-server می باشد. معماری client-server

یک معماری رایج برای طراحی وب سایت و برنامه‌های کاربردی مبتنی بر وب می باشد که سمت client وظیفه‌ی ارتباط با کاربر و

سمت server وظیفه پردازش اطلاعات را بر عهده دارد.

فرایند کلی برنامه به این شکل است که کاربر زبان موردنظر خود را انتخاب می کند و صدای خود را ارسال می کند و سپس یک

پاسخ در قالب صوت دریافت می کند.

زبان های پشتیبانی شده عبارتند از:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| زبان | فرمت | زبان | فرمت | زبان | فرمت | زبان | فرمت |
| Arabic | AR | African | AF | English | EN | Persian | FA |
| Welsh | CY | Czech | CS | Catalan | CA | Bosnian | BS |
| Spanish | ES | Estonian | ET | Germany | DR | Danish | DA |
| Hindi | HI | French | FR | Finnish | FI | Greek | EL |
| Indonesian | ID | Italian | IT | Armenian | HY | Hungarian | HU |
| Polish | PL | Korean | KO | Kannada | KN | Japanese | JA |
| Latvian | LV | Latin | LA | Portuguese | PT | Malay | ML |
| Dutch | NL | Nepali | NE | Marathi | MR | Macedonian | MK |
| Turkish | TR | Romanian | RO | Russian | RU | Norwegian | NO |
| Urdu | UR | Swedish | SV | Slovak | SK | Serbian | SR |
|  |  |  |  |  |  | Chinese | ZH |

در ادامه به بررسی این دو بخش در پروژه موردنظر و تکنولوژی های به کار رفته در آن می پردازیم:

**قسمت server:**

تکنولوژی‌های به کار رفته در سمت سرور عبارتند از:

1.**تبدیل صوت به متن**:

تبدیل صوت به متن فرایندی است که در آن به کمک الگوریتم‌های پیچیده و شبکه‌های عصبی آموزش دیده شده صدای کاربر

تشخیص داده می شود و تبدیل به متن می شود و چون هدف این برنامه تعامل صوتی کاربر با ChatGPT می‌باشد بنابراین باید از

این تکنولوژی استفاده کنیم.

برای آن که بتوانیم صدا‌ی کاربر را به متن تبدیل کنیم از کتابخانه‌ی OpenAI و از ماژول whisper استفاده می کنیم.

تکنولوژی whisper یکی از جدیدترین مدل های تبدیل صوت به متن می باشد که توسط شرکت OpenAI توسعه داده شده

است. این مدل از اکثر زبان‌های انسانی و به خصوص زبان فارسی پشتیبانی می کند که علت اصلی استفاده از کتابخانه همین بوده

است. این کتابخانه از 7 فرمت صوتی پشتیبانی می‌کند که این فرمت ها عبارتند از:

mp3, mp4, mpeg, mpga, m4a, wav, webm

2.**تبدیل متن به صوت:**

در این تکنولوژی هدف ، این است که به کمک الگوریتم‌های پیچیده‌ی هوش مصنوعی ، یک صدای مصنوعی برای خواندن یک

متن را فراهم کنیم. از این تکنولوژی در این پروژه برای این استفاده می شود که جواب ChatGPT را به صورت صوتی به کاربر

بدهیم.

کتابخانه های و API های مختلفی به صورت رایگان و پولی برای انجام این کار وجود دارد که در نهایت 2 کتابخانه برای این کار

انتخاب شده است:

الف)gTTS: سرویس تبدیل متن به صوت گوگل که برای زبان پایتون ارائه شده است اما متاسفانه از زبان فارسی پشتیبانی نمی‌کند.

ب)سرویس ابری اریانا: برای پر کردن خلأ زبان فارسی از سرویس پولی اریانا استفاده می کنیم.

نکته: برای استفاده از این سرویس باید به سایت <https://asr-gooyesh.com/fa> مراجعه کنید و اقدام به ساخت اکانت و

خرید سرویس مورد‌نظر خود بکنید

3.**زبان برنامه نویسی پایتون:**

قسمت سرور پروژه به کمک زبان پایتون پیاده سازی شده است . علت استفاده از پایتون ، پشتیبانی بالای کتابخانه‌های تبدیل متن

به صوت و صوت به متن از پایتون می باشد.

**قسمت کلاینت:**

1.کتابخانه‌ی React: یکی از محبوب ترین کتابخانه های موجود برای طراحی رابط کاربری وب کتابخانه‌ی React می باشد.

این کتابخانه بر پایه Component ها می‌باشد برنامه نویسان می توانند با ساخت کامپوننت‌های مستقل مانند کامپوننت دکمه ،

ضبط صدا ، دریافت ورودی و ... کامپوننت‌های ترکیبی بسازند و با ترکیب این کامپوننت های ترکیبی می شود صفحات مختلف

یک وبسایت را ساخت.

2.فریمورک Remix: این فریمورک که بر پایه‌ی کتابخانه‌ی React ساخته شده است استفاده از React با فراهم کردن قابلیت

هایی مانند مسیریابی ، بهینه سازی و ... تسهیل کرده است.

3.کتابخانه‌ی Tailwindcss: این کتابخانه یکی از محبوب‌ترین کتابخانه‌های موجود برای استایل‌دهی به عناصر HTML می‌باشد

که امکان استایل‌دهی را در کنار تعریف عانصر HTML فراهم می‌کند.

**پروتکل ارتباطی کلاینت با سرور:**

از زمانی که کاربر صدای خود‌ را ضبط و ارسال می کند تا زمانی که نتیجه را دریافت می‌کند و به آن گوش می‌دهد زمان زیادی

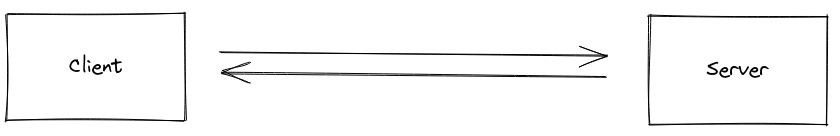
طول می‌کشد؛ علت آن واضح است چون که فرایند اپلود و دانلود و همچنین تبدیل متن به صوت و برعکس زمان‌بر است و برای

بهبود تجربه کاربری باید این زمان را کاهش داد بنابراین به جای استفاده از بستر ارتباطی REST API از تکنولوژی سوکت استفاده

شده است.

برای این منظور در سمت کلاینت و سرور باید از کتابخانه‌های مناسبی استفاده کرد. کتابخانه‌ی socketIO انتخاب مناسبی برای

این کار می باشد.



**پیاده سازی پروژه:**

**سمت سرور:**

کتابخانه‌های مورد نیاز برای تعامل صوتی با ChatGPT

* eventlet
* socketio
* openai
* revChatGPT
* gTTS

ماژول های پیاده سازی شده لازم برای پروژه

* tts(text to speech)
* stt(speech to text)

در ابتدای شروع پروژه لازم است که این کتابخانه تنظیم شوند.

رخداد‌های تعریف شده برای سوکت

* connect
* disconnect
* query

دو رخداد connect و disconnect صرفا برای اهداف توسعه‌ی برنامه می باشد و عملکرد اصلی برنامه در رخداد query

تعریف شده است که به شرح زیر می باشد.

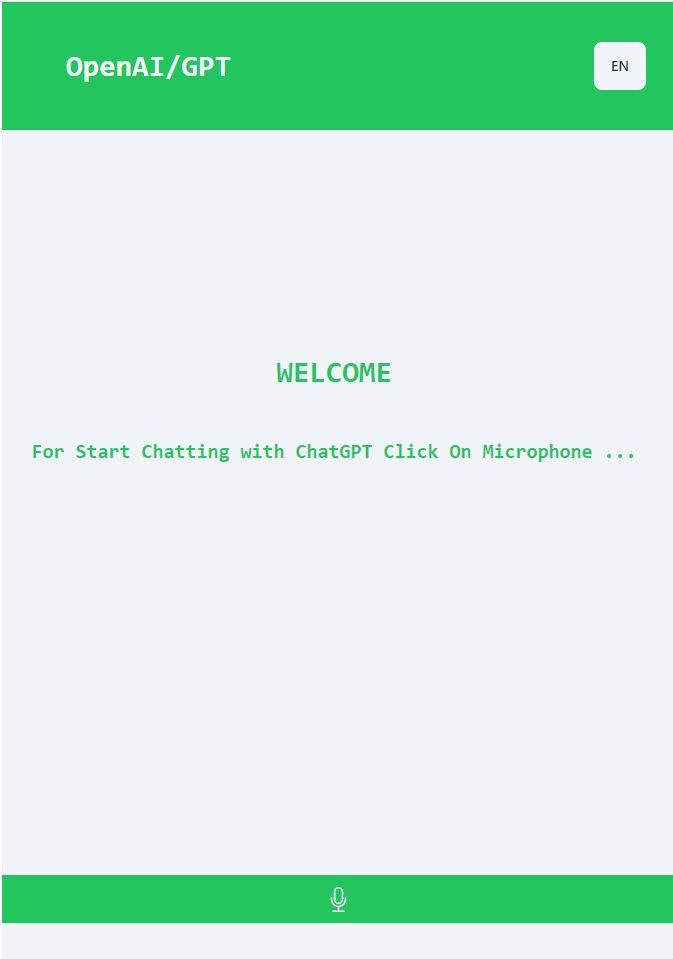
1. ذخیره‌ی بافر دریافتی در قالب یک فایل صوتی
2. فراخوانی ماژول stt روی فایل صوتی کوئری و دریافت نتیجه
3. ارتباط با ChatGPT
4. فراخوانی ماژول tts روی متن دریافتی و دخیره‌ی نتیجه
5. ارسال نتیجه به سمت کلاینت

**سمت کلاینت**

صفحات تعریف شده:

* صفحه‌ی index : تمام تعامل با ChatGPT در همین صفحه رخ می دهد.

در صفحه‌ی بعد می توانید یک نما (شروع اغازین) از صفحه‌ی index را مشاهده بکنید.



کامپوننت های تعریف شده:

* audio-recorder: تعریف شده برای ضبط صدا

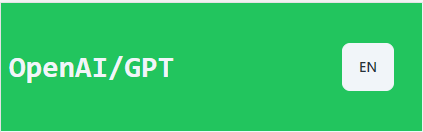




* audio: تعریف شده برای نمایش صداها



* language-selector: تعریف شده برای انتخاب زبان توسط کاربر



کاربر با کلیک روی دکمه‌ی EN می تواند زبان خود را تغییر دهد(این دکمه در هدر سایت قرار گرفته است).

* message: تعریف شده برای نمایش پیام‌ها بین کاربر و چت‌بات



کاربر با کلیک بر روی علامت میکروفون می تواند صدای خود را ضبط کند و پس از ضبط صدا با دکمه‌ی send در گوشه‌ی پایین

سمت راست می‌تواند آن را به سمت سرور ارسال کند و بعد از چند ثانیه جواب خود را دریافت کند. همچنین کاربر می تواند برای

هر سوال زبان خود را تغییر دهد.

**راه اندازی پروژه**

همانطور که گفته شد پروژه از دو برنامه‌ی مجزا شامل سمت سرور و سمت کلاینت تشکیل شده است که در ادامه به شرح هر دو

خواهیم پرداخت.

پیش نیاز: دانلود پروژه از لینک گیتهاب

<https://github.com/>

پروژه شامل 2 برنامه مجزا می باشد که می توان آن ها را هم در محیط توسعه (برای تغییر کد و توسعه‌ی آن) و هم در محیط

تولید اجرا کرد

**سمت سرور**

محیط توسعه:

برای توسعه‎ی محصول در کامپیوتر شخصی خود نیاز به نصب موارد زیر دارید:

1.

محیط تولید:

**سمت سروکلاینت**

محیط توسعه:

محیط تولید:

**ضمیمه کدها:**

**سمت سرور**

فایل app.py (نقطه‌ی شروع):

- import کردن کتابخانه‌ها:

import **eventlet**

import **socketio**

import **openai**

from **uuid** import **uuid4**

from **revChatGPT**.**V3** import **Chatbot**

from **dotenv** import **load\_dotenv**

import **os**

from **tts**.**text\_to\_speech** import **text\_to\_speech**

from **stt**.**speech\_to\_text** import **speech\_to\_text**

- مقدار دهی اولیه و تنظیمات:

*#init application*

**load\_dotenv**()

**openai**.api\_key = **os**.environ['OPENAI\_KEY']

api\_key = **os**.environ['OPENAI\_KEY']

ariana\_key = **os**.environ['ARIANA\_KEY']

FILE\_STORE\_PATH = **os**.environ['DATA\_PATH']

APP\_IP = **os**.environ['APP\_IP']

APP\_PORT = **os**.environ['APP\_PORT']

server = **socketio**.**Server**(cors\_allowed\_origins='\*',max\_http\_buffer\_size=100000000)

app = **socketio**.**WSGIApp**(server)

- رخدادهای connect و disconnect (برای اهداف توسعه):

**@server.event**

def **connect**(\_,\_\_):

**print**("start connecting ...")

**@server.event**

def **disconnect**(\_):

**print**("disconnecting ...")

- رخداد query:

**@server.event**

def **query**(id,data):

    response = {

        "status":False,

        "data":"Try Again ..."

    }

    filename = **uuid4**()

    query\_file\_path =  '{path}{filename}-query.wav'.**format**( path=FILE\_STORE\_PATH,filename=filename)

    answer\_file\_path = '{path}{filename}-answer.mp3'.**format**(path=FILE\_STORE\_PATH,filename=filename)

*#save query file in memory*

    try:

        with **open**(query\_file\_path,mode='bx') as file:

            file.**write**(data["audio"])

    except **Exception** as e:

        response["data"] = "Your voice is not clear, please speak more clearly"

        server.**emit**('answer',response,to=**id**)

        return

    try:

*#read file and convert it to text*

        query\_text = **speech\_to\_text**(query\_file\_path,data["lang"])

**print**(query\_text)

    except **Exception** as e:

        server.**emit**('answer',response,to=**id**)

        return

*#chat with chatGPT*

    try:

        chatbot = **Chatbot**(api\_key=api\_key)

        ai\_res\_text = chatbot.**ask**('please answer my question in lnaguage {lang} . {q}'.**format**(lang=data["lang"],q=query\_text))

**print**('answer :: {a}'.**format**(a=ai\_res\_text))

        ai\_res\_is\_successful = True

    except **Exception** as e:

**print**(e)

        ai\_res\_text = **str**(e)

        ai\_res\_is\_successful = False

    if(ai\_res\_is\_successful == False):

        server.**emit**('answer',response,to=**id**)

        return

    try:

**text\_to\_speech**(0,3,ai\_res\_text,data["lang"],answer\_file\_path,ariana\_key)

*#send answer file to client*

        f = **open**(answer\_file\_path,mode="rb")

        answer\_audio\_mp3\_file = f.**read**()

        f.**close**()

        response = {

            "status":True,

            "data":answer\_audio\_mp3\_file

        }

        server.**emit**('answer',response,to=**id**)

    except **Exception** as e:

        server.**emit**('answer',response,to=**id**)

        return

    try:

**os**.**remove**(query\_file\_path)

**os**.**remove**(answer\_file\_path)

    except **Exception** as e:

**print**(**str**(e))

    return

انتهای فایل به منظور شروع برنامه :

if \_\_name\_\_ == "\_\_main\_\_":

**eventlet**.wsgi.server(**eventlet**.listen((APP\_IP,**int**(APP\_PORT))),app)

فایل text\_to\_speech.py در ماژول tts:

from **gtts** import **gTTS**

import **requests**

def **text\_to\_speech**(index,limit,text,lang,file\_path,ariana\_key):

    try:

        if(lang == "fa"):

*#convert text to speech*

            r = **requests**.**post**("http://api.farsireader.com/ArianaCloudService/ReadText",json={

                "Text" : text,

                "Speaker":"Male1",

                "Quality":"normal",

                "Format":"mp3",

                "APIKey":ariana\_key

            },headers={'Content-type':'application/json'})

            with **open**(file\_path,mode='wb') as file:

                file.**write**(r.content)

        else:

*#convert text to speech*

            answer\_audio = **gTTS**(text=text,lang=lang,slow=False)

            answer\_audio.**save**(file\_path)

    except **Exception** as e:

        if(index >= limit) :

            raise e

        else:

**text\_to\_speech**(index+1,limit,text,lang,file\_path)

فایل speech\_to\_text در ماژول stt:

import **openai**

def **speech\_to\_text**(text\_path,language):

    x = **open**(text\_path,"rb")

    query\_text = **openai**.**Audio**.**transcribe**("whisper-1",x,language)["text"]

    return query\_text

فایل requirements.txt:

eventlet==0.33.3

python-engineio==4.3.4

python-socketio==5.7.2

revChatGPT==4.0.6.1

openai==0.27.2

gTTS==2.3.1

فایل Dockerfile

FROM **python**:3.10-alpine

RUN apt-get update && apt-get install build-essential -y

WORKDIR /usr/src/app

COPY requirements.txt requirements.txt

RUN pip install --no-cache-dir --upgrade pip && **\**

    pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["python","app.py"]

فرمت فایل .env

OPENAI\_KEY=your-openai-secret-key

ARIANA\_KEY=your-ariana-secret-key

DATA\_PATH=a relative or absolute path

APP\_PORT=8000

APP\_IP=0.0.0.0

**سمت کلاینت:**

فایل package.json:

{

  "private": true,

  "sideEffects": false,

  "scripts": {

    "build": "remix build",

    "dev": "remix dev",

    "start": "remix-serve build",

    "typecheck": "tsc"

  },

  "dependencies": {

    "@remix-run/node": "^1.13.0",

    "@remix-run/react": "^1.13.0",

    "@remix-run/serve": "^1.13.0",

    "@types/recordrtc": "^5.6.10",

    "daisyui": "^2.51.2",

    "isbot": "^3.6.5",

    "react": "^18.2.0",

    "react-dom": "^18.2.0",

    "react-hot-toast": "^2.4.0",

    "react-icons": "^4.7.1",

    "react-media-recorder": "^1.6.6",

    "react-textarea-autosize": "^8.4.0",

    "react-use": "^17.4.0",

    "recordrtc": "^5.6.2",

    "remix-utils": "^6.0.0",

    "socket.io-client": "^4.6.1"

  },

  "devDependencies": {

    "@remix-run/dev": "^1.13.0",

    "@remix-run/eslint-config": "^1.13.0",

    "@types/react": "^18.0.25",

    "@types/react-dom": "^18.0.8",

    "eslint": "^8.27.0",

    "tailwindcss": "^3.2.7",

    "typescript": "^4.8.4"

  },

  "engines": {

    "node": ">=14"

  }

}

صفحه‌ی index:

import {ClientOnly} from 'remix-utils';

import { Message } from "~/components/message";

import {AudioRecorder} from "~/components/audio-recorder3.client";

import { useVMContext } from '~/voice-memory';

import { useState } from 'react';

import LanguageSelector from '~/components/language-selector';

export default function **Index**() {

  const {voices} = **useVMContext**();

  const [lang,setLang] = **useState**<string>('en');

  return (

    <div *className*="flex flex-col w-full bg-primary lg:w-[55%] xl:w-[35%] mx-auto min-h-screen max-h-screen h-screen">

      <header *className*="flex flex-row w-full h-32 bg-secondary">

        <span *className*="my-auto ml-2 lg:ml-16 text-primary font-mono font-extrabold text-3xl">

          OpenAI/GPT

        </span>

        <div *className*="my-auto ml-auto mr-6">

          <**LanguageSelector** *lang*={lang} *setLang*={setLang} />

        </div>

      </header>

      <div *className*="w-full msg-container p-4 max-h-[calc(100%-14rem)] h-[calc(100%-14rem)] overflow-y-scroll flex flex-col">

      {

        voices.length === 0 ?

        <div *className*="text-secondary font-mono font-extrabold flex flex-col">

          <p *className*='text-3xl mx-auto mt-52'>WELCOME</p>

          <p *className*='text-xl  mx-auto mt-12'>For Start Chatting with ChatGPT Click On Microphone ...</p>

        </div>

        :

        <>

          {

            voices.**map**((voice)=>

              <div *key*={voice.id} *className*="my-4">

                <**Message**

*msgOwnerUsername*={voice.sender}

*voiceSrc*={voice.blobUrl}

                />

              </div>

            )

          }

        </>

      }

      </div>

      <**ClientOnly** *fallback*={<p>loading .... </p>}>

        {

          ()=> <**AudioRecorder** *lang*={lang}/>

        }

      </**ClientOnly**>

    </div>

  );

}

کامپوننت ضبط صدا:

import { AiOutlineDelete } from 'react-icons/ai';

import { TfiControlPause } from 'react-icons/tfi';

import { BsMic } from 'react-icons/bs';

import { useCallback, useState } from 'react';

import Audio from '~/components/audio.client';

import {AiOutlineSend} from 'react-icons/ai';

import useAudioRecorder from '~/audio-recorder.client';

import { useVMContext } from '~/voice-memory';

import { useChat } from '~/socket';

export function **AudioRecorder**({lang}:{lang:string}) {

    const [blobUrl , **setBlobUrl**] = **useState**<string>('');

    const [**blob**,**setBlob**] = **useState**<**Blob**|null>();

    const {voices,setVoices} = **useVMContext**();

    const {**sendVoiceMessage**} = **useChat**({setVoices,voices});

    const [micState, **setMicState**] = **useState**<string>('record');

    const {**start**,**stop**} = **useAudioRecorder**({

**onStart**() {

**setMicState**("recording");

        },

**onStop**:**useCallback**((blob:**Blob**,blobUrl:string)=>{

**setMicState**("recorded");

**setBlob**(blob);

**setBlobUrl**(blobUrl);

        },[]),

    });

    const **handleSendVoice** = () => {

        if(**blob** === undefined || **blob** === null) return;

**sendVoiceMessage**(**blob**,lang);

**setBlob**(null);

**setBlobUrl**("");

**setMicState**("record");

    };

    const **handleDelete** = () => {

**setMicState**("record");

**setBlob**(null);

**setBlobUrl**("");

    };

    if (micState === "record") {

        return (

            <div *className*="flex flex-row w-full h-12 bg-secondary text-black">

                <**BsMic** *onClick*={**start**} *size*={25} *className*="fill-primary hover:fill-third m-auto" />

            </div>

        )

    } else if (micState === "recording") {

        return (

            <div *className*="flex flex-row w-full h-12 bg-secondary text-black">

                <**TfiControlPause** *onClick*={**stop**} *size*={25} *className*="fill-primary hover:fill-third m-auto" />

            </div>

        )

    } else{

*// else if (micState === "recorded")*

        return (

            <div *className*="flex flex-row w-full h-12 bg-secondary text-black">

                <div *className*="my-auto">

                    <div *className*="h-8 w-8 ml-0 sm:ml-52 lg:ml-16 2xl:ml-20">

                        <**AiOutlineDelete** *onClick*={**handleDelete**} *size*={25} *className*="fill-primary hover:fill-third" />

                    </div>

                </div>

                <div *className*='my-auto h-8 w-52'>

                    <**Audio** *src*={blobUrl} />

                </div>

                <div *className*="my-auto ml-auto">

                    <div *className*="h-8 w-8">

                        <**AiOutlineSend** *onClick*={**handleSendVoice**} *size*={25} *className*="m-auto fill-primary hover:fill-third" />

                    </div>

                </div>

            </div>

        )

    }

}

کامپوننت صدا به همراه وابستگی ها:

import {BsPlay} from 'react-icons/bs';

import {TfiControlPause} from 'react-icons/tfi';

import { useAudio } from 'react-use';

import { ClientOnly } from 'remix-utils';

export default function **Audio**({src}:**AudioProps**){

    const [audio,state,controls] = **useAudio**({src});

    return (

        <**ClientOnly** *fallback*={<p>loading</p>}>

        {

          () =>

            <div *className*="flex flex-row">

                <**TimeFormat**  *time*={state.duration}/>:<**TimeFormat** *time*={state.time}/>

            {

                state.paused ?

                <button

*onClick*={()=>controls.**play**()}

                ><**BsPlay** *size*={25} *className*="fill-primary hover:fill-third"/></button>

                :

                <button

*onClick*={()=>controls.**pause**()}

                ><**TfiControlPause** *size*={25} *className*="fill-primary hover:fill-third"/></button>

            }

            {audio}

            <**AudioBar** *seek*={controls.**seek**} *time*={state.time} *duration*={state.duration} *className*="mx-2 mt-[6px]"/>

            </div>

        }

        </**ClientOnly**>

    );

}

interface **AudioProps**{

    src:string;

}

function **AudioBar**({

    className,time,duration,**seek**

}:**AudioBarProps**){

*// useEffect(()=>{*

*//     //this is a hack for fix chrome bug*

*//     if(duration===Infinity){*

*//         seek(15000);*

*//         seek(0);*

*//     }*

*// },[duration,seek]);*

    const **handleChange** = (e:any) =>{

        try{

            const valNum = **parseFloat**(e.target.value);

**seek**(valNum/100\*duration );

        }catch(e){}

    }

    return(

        <div *className*={`${className}`}>

            <input

*type*="range"

*step*="any"

*onChange*={**handleChange**}

*value*={`${duration !== Infinity? time/duration\*100:time/50\*100}`}

*className*="range range-xs range-primary w-16 min-[335px]:w-24 min-[380px]:w-36 "

            />

        </div>

    )

}

interface **AudioBarProps**{

    className?:string;

    time:number;

    duration:number;

**seek**:(time:number)=>void;

}

function **TimeFormat**({time}:{time:number}){

    if(time === Infinity){

        return(

            <span *className*="pt-0.5">0:00</span>

        )

    }

    return(

        <span *className*="pt-0.5">

        {Math.**floor**(time/60)}

        :

        {Math.**floor**(time) - (Math.**floor**(time / 60)) \* 60 < 10 ? 0 : ''}{Math.**floor**(time) - (Math.**floor**(time / 60)) \* 60}

        </span>

    )

}

کامپوننت انتخاب زبان:

const langs = [

    "fa","en",

    "ar","af",

    "bs",

    "ca","cs","cy",

    "da","dr",

    "el","et","es",

    "fi","fr",

    "hy","hu","hi",

    "it","id",

    "kn","ko","ja",

    "la","lv",

    "ml","mr","mk",

    "nl","ne","no",

    "ru","ro",

    "sr","sk","sv",

    "tr",

    "pl","pt",

    "ur",

    "zh",

];

interface **LSProps**{

    lang:string;

    setLang:any;

}

export default function **LanguageSelector**({lang,setLang}:**LSProps**){

    return(

        <div *className*="dropdown text-secondary">

            <label *className*="btn btn-primary m-1" *tabIndex*={0}>{lang}</label>

            <ul *tabIndex*={0}

*className*="msg-container dropdown-content menu shadow bg-primary rounded-sm w-16 h-40 overflow-y-auto overflow-x-hidden flex flex-row scrol"

            >

                {

                    langs.**map**(l=>

                    <li *key*={l} *onClick*={()=>**setLang**(l)}>

                        <a>{l}</a>

                    </li>)

                }

            </ul>

        </div>

    )

}

کامپوننت پیام:

import {BsPerson} from 'react-icons/bs';

import {AiOutlineRobot} from 'react-icons/ai';

import Audio from '~/components/audio.client';

interface **MessageProps**{

    msgOwnerUsername:string;

    voiceSrc:string;

}

export function **Message**(

    {

        msgOwnerUsername,

        voiceSrc

    }:**MessageProps**

){

    const username = "1";

    return(

        <div *className*={`chat ${username === msgOwnerUsername ? "chat-start":"chat-end"} `}>

            <div *className*="chat-image avatar ">

                <div *className*=" rounded-full">

                    {

                        username === msgOwnerUsername ?

                        <**BsPerson** *size*={45} *className*="fill-secondary"/>

                        :

                        <**AiOutlineRobot** *size*={45} *className*="fill-secondary"/>

                    }

                </div>

            </div>

            <div *className*='chat-header font-mono text-xl text-secondary'>

            {username === msgOwnerUsername?"You":"GPT"}

            </div>

            <div *className*='chat-bubble  bg-secondary text-primary'>

                <**Audio** *src*={voiceSrc}/>

            </div>

        </div>

    )

}

منطق مربوط به سوکت (فایل socket.ts):

import { useContext,createContext, useEffect, useState } from "react";

import type {Socket} from 'socket.io-client';

import {io} from 'socket.io-client';

import type { VMType, Voice } from './voice-memory';

import {toast} from 'react-hot-toast';

export const SocketContext = **createContext**<**Socket**|undefined>(undefined);

export function **useSocketContext**(){

    return **useContext**(SocketContext);

}

export function **useSocket**(){

    const [socket,**setSocket**] = **useState**<**Socket**>();

**useEffect**(()=>{

        const socket = **io**(`${window.ENV.SERVER\_IP}:${window.ENV.SERVER\_PORT}/`);

*// const socket = io("http://localhost:8000/");*

**setSocket**(socket);

        return ()=>{

            socket.**close**();

        };

    },[]);

    return socket;

}

export function **useChat**(vm\_handler:**VMType**){

    const socket = **useSocketContext**();

**useEffect**(()=>{

        if(socket === undefined) return;

        socket.**on**("answer",(data)=>{

            const status = data["status"];

            if(status === false){

                toast.**error**("Try Again ...");

                return;

            }

            const buffer = data["data"];

            console.**log**(data)

*//data is arrayBuffer.*

*//i convert to blob*

            const **blob** = new **Blob**([buffer]);

            const bloburl = **URL**.**createObjectURL**(**blob**);

            vm\_handler.**setVoices**((voices:**Voice**[])=>{

                const v:**Voice** = {blobUrl:bloburl,id:'0',sender:'0'};

                return [...voices,v]

            });

        });

        return () => {

            socket.**off**("answer");

        }

    },[socket,vm\_handler]);

    const **sendVoiceMessage** = (voice:**Blob**,lang:string) => {

        if(socket === undefined) return;

        const bloburl = **URL**.**createObjectURL**(voice);

        vm\_handler.**setVoices**((voices:**Voice**[])=>{

            const v:**Voice** = {blobUrl:bloburl,id:'1',sender:'1'};

            return [...voices,v]

        });

*// const file = new File([voice],"v.wav");*

        console.**log**(voice)

        socket.**emit**("query",{**audio**:voice,lang});

    };

    return {**sendVoiceMessage**};

}

مراجع و منابع:

1. <https://remix.run/docs/en/main>

2. <https://socket.io/>

3. <https://python-socketio.readthedocs.io/en/latest/>

4. <https://tailwindcss.com/>

5. <https://asr-gooyesh.com/fa/>

6. <http://farsireader.com/>

7. <https://openai.com/>