**بسم الله الرحمن الرحیم**

پروژه‌ی دوره‌ی کارشناسی

موضوع: گزارش پروژه‌ی پیاده سازی چت صوتی با ChatGPT

استاد راهنما : سرکار خانم دکتر بهکمال

توسعه دهنده : رضا نصیری

فهرست

مقدمه 3

معماری پروژه 3

- سمت سرور 4

-سمت کلاینت 5

پیاده سازی پروژه 6

- سمت سرور 6

-سمت کلاینت 7

راه اندازی پروژه 12

- سمت سرور 12

-سمت کلاینت 16

ضمیمه 18

- سمت سرور 18

-سمت کلاینت 24

منابع 35

**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 توانایی‌های آن 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 سازی‌ای که 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.

**معماری 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

whisper که یک سیستم تشخیص گفتار می‌باشد برروی 680000 ساعت داده های چند زبانه آموزش داده شده است. استفاده

از چنین حجم بزرگی از داده ها منجر به بهبود لهجه‌های گوناگون و رفع نویزهای زبانی بشود.

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:

سروس gTTS که مخفف google text to speech می‌باشد سرویس تبدیل متن به صوت گوگل که برای زبان برنامه نویسی

پایتون ارائه شده است و از اکثر زبان های دنیا مانند عربی ،انگلیسی ، آلمانی ، فرانسوی پشتیبانی می کند اما متاسفانه از زبان

فارسی پشتیبانی نمی‌کند.

ب)سرویس ابری اریانا:

سرویس ابری اریانا یک سرویس ایرانی می باشد که برای تبدیل متن فارسی به صوت فارسی کاربرد دارد و ما از آن برای پر کردن

خلأ زبان فارسی در gTTS از سرویس 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:**

پایتون یکی از زبان های برنامه نویسی محبوب می باشد در حوزه‌های مختلفی مانند پردازش تصویر ، هوش مصنوعی ، اینترنت

اشیا ، برنامه نویسی سرور ، اسکریپت نویسی و ... کاربرد دارد.

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 قابلیت‌هایی 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 عناصر 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

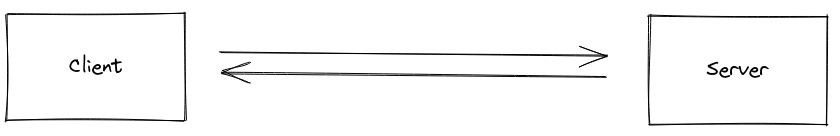
طول می‌کشد؛ علت آن واضح است چون که فرایند اپلود و دانلود و همچنین تبدیل متن به صوت و برعکس زمان‌بر است و برای

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)

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 از طریق کتابخانه‌ی مرتبطه
4. فراخوانی ماژول tts روی متن دریافتی و دخیره‌ی نتیجه
5. ارسال نتیجه به سمت کلاینت

**سمت کلاینت**

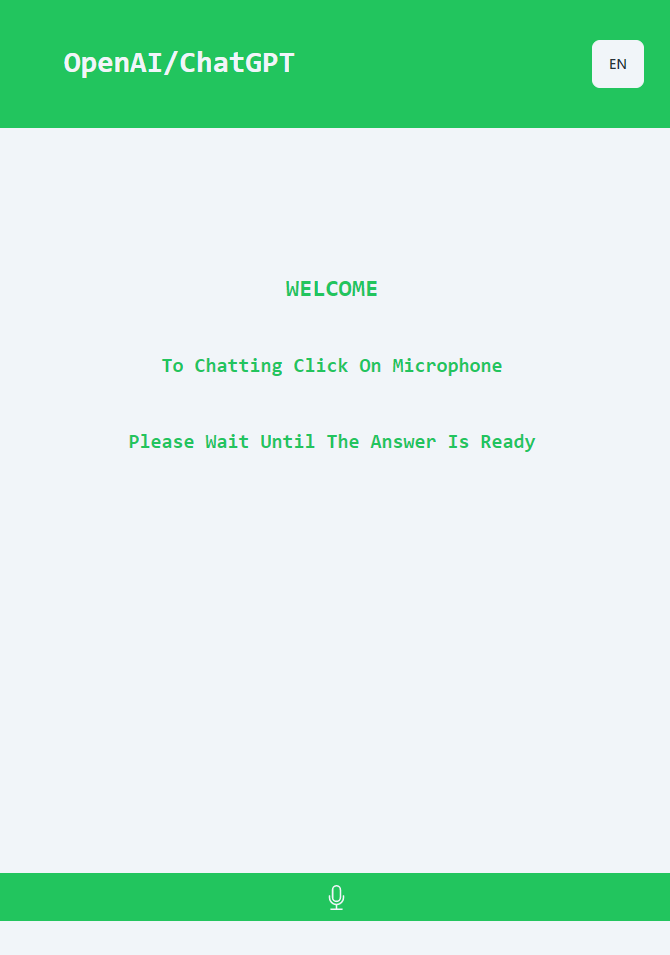
صفحات تعریف شده:

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

این تعامل شامل موارد زیر می باشد:

1. ضبط صدا
2. ارسال سوال
3. تغییر زبان
4. دریافت نتیجه
5. گوش کردن به صداها
6. توانایی عقب و جلو رفتن در صداها

در ادامه 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

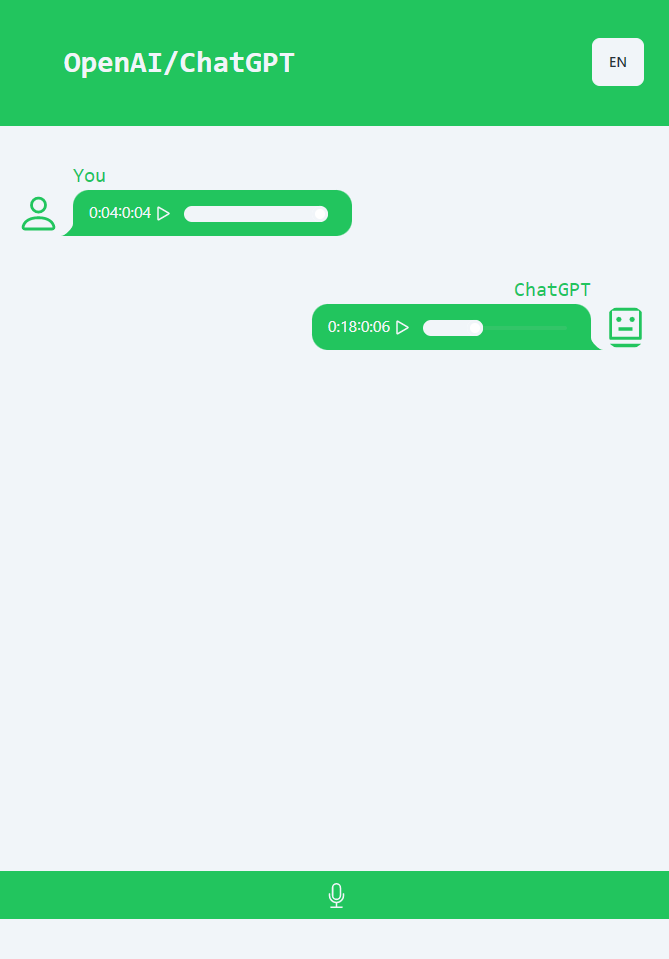


کاربر با کلیک بر روی علامت میکروفون می تواند صدای خود را ضبط کند و پس از ضبط صدا با دکمه‌ی send در گوشه‌ی پایین

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.

تصویر زیر نیز این صفحه را با تمام اجزایش پس از پرسیدن یک سوال نشان می‌دهد.



منطق قسمت‌های مختلف برنامه:

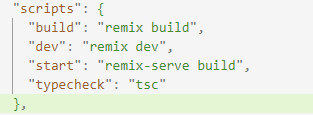
* ضبط صدا
  + شروع:
    - استفاده از mediaDevices مرورگر و شروع به ضبط صدای کاربر
    - استفاده از AudioRecorder برای capture کردن صدای کاربر به منظور پردازش آن
  + توقف:
    - توقف AudioRecorder و گرفتن خروجی از آن
    - پاک کردن ترک های mediaDevices
* ارسال و دریافت صدا از سرور
  + ارسال:
    - ساخت blobURL برای blob موجود
    - ذخیره کردن آن در حافظه
    - ارسال کردن به سمت سرور از طریق رخداد query
  + دریافت:
    - استخراج بافر صدا از داده‌ی ارسال شده
    - ساخت blob از آن
    - ساخت blobURL از آن
    - ذخیره‌ی آن ها

تنظیمات قسمت های مختلف برنامه:

* فایل package.json
  + مشخصات پروژه



* + وابستگی های پروژه
  + اسکریپت های لازم برای اجرا و ساخت



* فایل remix.config.js
  + پورت توسعه‌ی برنامه
  + استفاده یا عدم استفاده از tailwindcss
* فایل tailwindcss.confgi.js
  + وارد کردن کتابخانه های براساس tailwindcss
  + مشخص کردن رنگ های اصلی برنامه



* فایل tsconfig.json
  + تنظیمات لازم برای اضافه کردن تایپ اسکریپت به پروژه

**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 نیازها:

-دانلود پروژه از لینک گیتهاب که در زیر آمده است:

<https://github.com/>reza-n051/OpenAI-GPT

پوشه‌ی server شامل کد قسمت backend می باشد که شامل یک پروژه‌ی پایتون می باشد و پوشه‌ی front یک پروژه با

کتابخانه‌ی react می باشد.

**-**ثبت نام در سایت‌های openai و عصر گویش (سرویس اریانا) و گرفتن کلید های API

**سمت سرور:**

در ادامه نحوه‌ی راه اندازی به سه شیوه‌ی متفاوت توضیح دده می شود.

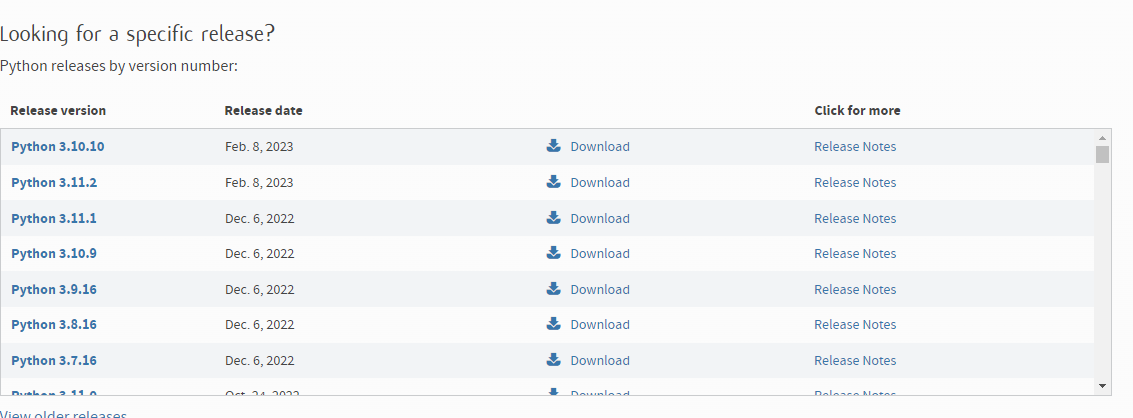
- روش معمولی (مناسب توسعه) :

نیاز است که ورژن نیازمندی ها برای عملکرد بهتر رعایت شود.

1.نصب پایتون (نسخه‌ی 3.10) و pip (نسخه‌ی 21.2.3)

website = <https://www.python.org/downloads>

برای نصب به سایت بالا مراجعه کرده و از قسمتی که طبق شکل زیر امده است پایتون را دانلود و نصب کنید.



2.نصب ماژول های موجود در فایل requirements.txt (شماره ورژن ها در این فایل مشخص شده اند)

eventlet==0.33.3

python-engineio==4.3.4

python-socketio==5.7.2

python-dotenv==0.20.0

revChatGPT==4.0.6.1

openai==0.27.2

gTTS==2.3.1

نحوه‌ی نصب هر کتابخانه به این شکل است که باید دستور زیر را وارد کنید:

pip install library

example: pip install openai

ممکن است که لازم باشد به جای دستور pip از دستور pip3 استفاده کنید.

pip3 install openai

بعد از نصب همه‎ی کتابخانه‌های لازم اماده رفتن به مرحله‌ی بعد هستید.

3.ساخت فایل .env در پوشه‌ی server و قراردادن موارد زیر در آن:

1. OPENAI\_KEY
2. ARIANA\_KEY
3. DATA\_PATH
4. APP\_IP
5. APP\_PORT

فرمت مقادیر در قسمت ضمیمه آمده است.

4.اجرای دستور زیر

python app.py

یا دستور

python3 app.py

در نهایت خط تصویر زیر را در ترمینال خود مشاهده خواهید کرد.



- به کمک داکر:

توجه شود که هدف از استفاده داکر راحت بودن صرفا تست برنامه می‌باشد و برای توسعه‌ی محصول از راه قبل

استفاده کرد و برای استقرار برنامه روی سرور واقعی باید به نکات امنیتی شامل متغیرهای محیطی توجه شود.

پیش نیاز : نصب داکر

1. ساخت فایل .env در پوشه‌ی server و قراردادن موارد زیر در آن:

1. OPENAI\_KEY
2. ARIANA\_KEY
3. DATA\_PATH
4. APP\_IP
5. APP\_PORT

فرمت مقادیر در قسمت ضمیمه آمده است.

2.وارد ترمینال شده و به پوشه‌ی server بروید.

3.دستورات زیررا اجرا کنید.

docker build -t server-app .

docker run --env-file .env -p 127.0.0.1:8000:8000 server-app

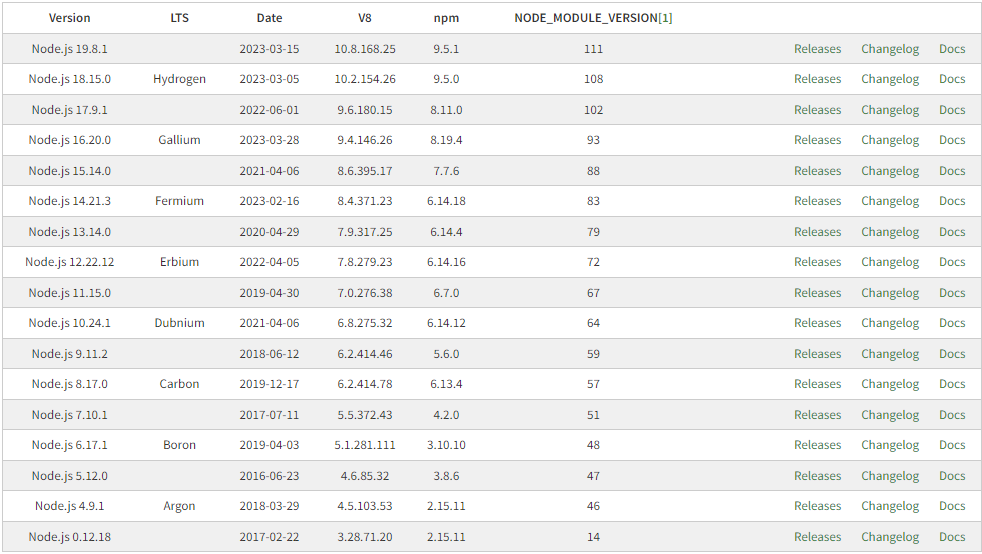
- به کمک پلتفرم لیارا:

این سرویس ایرانی امکان استقرار پروژه‌های برنامه نویسی را بدون نگرانی برای نکات مربوط به حوزه‌ی devops فراهم می کند.

website: <https://liara.ir>

1.ابتدا باید NodeJS را در کامپیوتر خود نصب کنید.

website = <https://nodejs.org/en/download/releases>



این عکس تمام نسخه‌های NodeJS را در حال حاضر نمایش می دهد. اگر در روند اجرای پروژه با مشکلی مواجه شدید نسخه‌ی

16 را نصب کنید.

2.ثبت نام در سایت لیارا

3.نصب liara-cli

برای این کار باید دستور زیر در ترمینال خود وارد کنید:

npm install -g @liara/cli

4. ساخت یک پلتفرم Docker از این سایت

برای آموزش ساخت به لینک زیر مراجعه کنید:

website = <https://docs.liara.ir/app-deploy/docker/getting-started>

5.بعد از مشاهده لینک بالا و ساخت پلتفرم داکر باید به مسیر پروژه بروید و دستور زیر را اجرا کنید:

liara deploy

در صورتی که به خطا برخورد کردید دستور زیر وارد کنید

npx liara deploy

6.مراجعه به سایت لیارا و وارد کردن متغیرهای محیطی :

1. OPENAI\_KEY
2. ARIANA\_KEY
3. DATA\_PATH
4. APP\_IP
5. APP\_PORT

فرمت مقادیر در قسمت ضمیمه آمده است.

نکته مهم 1: مقدار APP\_IP را برابر 0.0.0.0 قرار دهید.

نکته مهم 2: مقدار APP\_PORT را برابر شماره پورتی قرار دهید که هنگام اجرای دستور مرحله قبل وارد کردید.

**سمت کلاینت:**

- روش معمولی :

1.در ابتدا باید NodeJS را نصب کنید

2.بعد از اطمینان از نصب NodeJS وارد مسیر پروژه‌ی کلاینت بشوید.

3. ساخت فایل .env در پوشه‌ی front و قراردادن موارد زیر در آن:

1. SERVER\_PORT
2. SERVER\_IP

این مقادیر باید با توجه به IP و پورتی که برای پروژه سرور انتخاب کردید باشند.

فرمت مقادیر در قسمت ضمیمه آمده است.

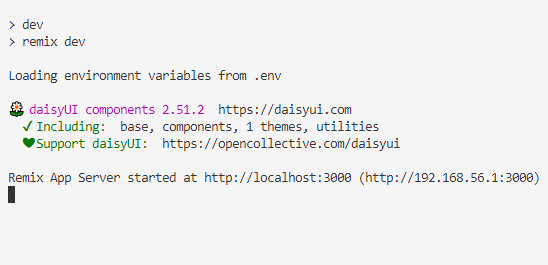
4.اجرای دستور زیر برای اجرای برنامه:

npm i

5.در نهایت با دستور زیر پروژه را اجرا کنید

npm run dev

پس از آنکه این دستور را اجرا کنید در ترمینال خود خطوط تصویر زیر‌ را مشاهده می‌کنید.



- به کمک پلفترم لیارا:

این سرویس ایرانی امکان استقرار پروژه‌های برنامه نویسی را بدون نگرانی برای نکات مربوط به حوزه‌ی devops فراهم می کند.

website: <https://liara.ir>

1.ابتدا باید NodeJS را در کامپیوتر خود نصب کنید.

website = <https://nodejs.org/en/download/releases>

2.ثبت نام در سایت لیارا

3.نصب liara-cli

برای این کار باید دستور زیر در ترمینال خود وارد کنید:

npm install -g @liara/cli

4. ساخت یک پلتفرم NodeJS از این سایت

برای آموزش ساخت به لینک زیر مراجعه کنید:

website = <https://docs.liara.ir/app-deploy/nodejs/getting-started>

5.بعد از مشاهده لینک بالا و ساخت پلتفرم داکر باید به مسیر پروژه بروید و دستور زیر را اجرا کنید:

liara deploy --port 3000 --platform node

در صورتی که به خطا برخورد کردید دستور زیر وارد کنید

npx liara deploy --port 3000 --platform node

6.مراجعه به سایت لیارا و وارد کردن متغیرهای محیطی :

1. SERVER\_IP
2. SERVER\_PORT

فرمت مقادیر در قسمت ضمیمه آمده است.

نکته مهم 1: مقدار SERVER\_IP را برابر ادرس دامنه‌ای قرار دهید که موقع ساخت پلتفرم داکر قرار داده‌ اید.

نکته مهم 2: مقدار SERVER\_PORT را برابر شماره پورتی قرار دهید که هنگام استقرار پلتفرم سرور وارد کردید.

**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["iso\_language"])

**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["text\_language"],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["iso\_language"],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**(

        model="whisper-1",

        file=x,

        language=language

    )["text"]

    return query\_text

فایل requirements.txt:

eventlet==0.33.3

python-engineio==4.3.4

python-socketio==5.7.2

python-dotenv==0.20.0

revChatGPT==4.0.6.1

openai==0.27.2

gTTS==2.3.1

فایل Dockerfile:

FROM **python**:3.10

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';

import { useLoadingContext } from '~/loading';

export default function **Index**() {

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

  const {isLoading} = **useLoadingContext**();

  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*="flex flex-col my-auto ml-2 lg:ml-16 text-primary font-mono font-extrabold text-3xl h-10">

          <span *className*="h-8">OpenAI/ChatGPT</span>

          {

            isLoading ? <progress *className*="progress progress-info bg-primary w-[100%] h-2"></progress> : <></>

          }

        </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-2xl mx-auto mt-32'>WELCOME</p>

          <p *className*='sm:text-xl  mx-auto mt-12'>To Chatting Click On Microphone</p>

          <p *className*='text-sm min-[380px]:text-base sm:text-xl  mx-auto mt-12'>Please Wait Until The Answer Is Ready</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:

import {langs} from '~/language-util';

interface **LSProps**{

    lang:string;

    setLang:any;

}

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

    return(

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

            <label *className*="btn btn-primary m-1 w-12" *tabIndex*={0}>{lang}</label>

            <ul *tabIndex*={0}

*className*="msg-container dropdown-content menu shadow bg-primary rounded-sm w-20 min-[358px]:w-28 h-40 overflow-y-auto overflow-x-hidden flex flex-row scrol"

            >

                {

                    langs.**map**(l=>

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

                        <a>{l.text}</a>

                    </li>)

                }

            </ul>

        </div>

    )

}

فایل language-util.ts

export const langs = [

    {iso:"fa",text:"persian"},

    {iso:"en",text:"english"},

    {iso:"ar",text:"arabic"},

    {iso:"af",text:"african"},

    {iso:"bs",text:"bosnian"},

    {iso:"ca",text:"catalan"},

    {iso:"cs",text:"Czech"},

    {iso:"cy",text:"Welsh"},

    {iso:"da",text:"danish"},

    {iso:"dr",text:"germany"},

    {iso:"et",text:"estonian"},

    {iso:"es",text:"spanish"},

    {iso:"el",text:"greek"},

    {iso:"fi",text:"finnish"},

    {iso:"fr",text:"french"},

    {iso:"hi",text:"hindi"},

    {iso:"hu",text:"hungarian"},

    {iso:"hy",text:"armanian"},

    {iso:"it",text:"italian"},

    {iso:"id",text:"indonesian"},

    {iso:"ja",text:"japanese"},

    {iso:"kn",text:"kannada"},

    {iso:"ko",text:"korean"},

    {iso:"pl",text:"polish"},

    {iso:"pt",text:"portuguese"},

    {iso:"la",text:"latin"},

    {iso:"lv",text:"latvian"},

    {iso:"ml",text:"malay"},

    {iso:"mk",text:"makedonain"},

    {iso:"mr",text:"marathi"},

    {iso:"ne",text:"nepali"},

    {iso:"nl",text:"dutch"},

    {iso:"no",text:"norwegian"},

    {iso:"ru",text:"russian"},

    {iso:"ro",text:"romanian"},

    {iso:"tr",text:"turkish"},

    {iso:"sr",text:"serbian"},

    {iso:"sk",text:"slovak"},

    {iso:"sv",text:"swedish"},

    {iso:"ur",text:"urdu"},

    {iso:"zh",text:"chinese"}

]

export function **isoToText**(iso:string = "en"){

    let res = "english";

    for(let i=0;i<langs.length;i++){

        if(iso === langs[i].iso) res = langs[i].text

    }

    return res;

}

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';

import { useLoadingContext } from "./loading";

import { isoToText } from "./language-util";

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

export function **useSocketContext**(){

    return **useContext**(SocketContext);

}

export function **useSocket**(){

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

**useEffect**(()=>{

        let socket:**Socket**;

        if(process.env.NODE\_ENV == "production"){

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

        }else{

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

        }

**setSocket**(socket);

        return ()=>{

            socket.**close**();

        };

    },[]);

    return socket;

}

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

    const socket = **useSocketContext**();

    const {setIsLoading}  = **useLoadingContext**();

**useEffect**(()=>{

        if(socket === undefined) return;

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

            const status = data["status"];

            if(status === false){

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

**setIsLoading**(false);

                return;

            }

            const buffer = data["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]

            });

**setIsLoading**(false);

        });

        return () => {

            socket.**off**("answer");

        }

    },[socket,vm\_handler]);

    const **sendVoiceMessage** = (voice:**Blob**,iso\_lang:string) => {

        if(socket === undefined) return;

**setIsLoading**(true);

        const text\_lang = **isoToText**(iso\_lang);

        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,iso\_language:iso\_lang,text\_language:text\_lang});

    };

    return {**sendVoiceMessage**};

}

مراجع:

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

2. [https://socket.io](https://socket.io/)

3. [https://python-socketio.readthedocs.io/en/latest](https://python-socketio.readthedocs.io/en/latest/)

4. [https://tailwindcss.com](https://tailwindcss.com/)

5. [https://asr-gooyesh.com/fa](https://asr-gooyesh.com/fa/)

6. [http://farsireader.com](http://farsireader.com/)

7. [https://openai.com](https://openai.com/)

8. [https://docs.liara.ir](https://docs.liara.ir/)