01 OPENAI + REACT + NEST

- El cascarón del front no lo he documentado
- No voy a explcar lo básico de NEST
- Iré directo al caso de uso (el primero es el de ortografía)

Backend sección 3

- Básicamente el proceso en todos los casos de uso va a ser el mismo
  - Crear el controlador respectivo
  - Crear el DTO
  - Llegar al caso de uso mediante el servicio
  - El caso de uso haga todo el trabajo
- En estas dos primeras secciones es donde está todo el contenido
- Las siguientes se centran solo en funciones

NOTA: OPENAI ha sacod un nuevo modelo GPT-4o. La ventaja es que usaremos el sdk propio de OPENAI por lo que el código es el mismo

---

Inicio del proyecto

- En la carpeta de React creo el proyecto

```
nest new nest-gpt
```

- Copio el package.json

```
{
  "name": "nest-gpt",
  "version": "0.0.1",
  "description": "",
  "author": "",
  "private": true,
  "license": "UNLICENSED",
  "scripts": {
    "build": "nest build",
    "format": "prettier --write \"src/**/*.ts\" \"test/**/*.ts\"",
    "start": "nest start",
    "start:dev": "nest start --watch",
    "start:debug": "nest start --debug --watch",
    "start:prod": "node dist/main",
    "lint": "eslint \"{src,apps,libs,test}/**/*.ts\" --fix",
    "test": "jest",
    "test:watch": "jest --watch",
```

```
    "test:cov": "jest --coverage",
    "test:debug": "node --inspect-brk -r tsconfig-paths/register -r ts-
node/register node_modules/.bin/jest --runInBand",
    "test:e2e": "jest --config ./test/jest-e2e.json"
  },
  "dependencies": {
    "@nestjs/common": "^10.0.0",
    "@nestjs/config": "^3.1.1",
    "@nestjs/core": "^10.0.0",
    "@nestjs/mapped-types": "*",
    "@nestjs/platform-express": "^10.0.0",
    "class-transformer": "^0.5.1",
    "class-validator": "^0.14.0",
    "openai": "^4.23.0",
    "reflect-metadata": "^0.1.13",
    "rxjs": "^7.8.1"
  },
  "devDependencies": {
    "@nestjs/cli": "^10.0.0",
    "@nestjs/schematics": "^10.0.0",
    "@nestjs/testing": "^10.0.0",
    "@types/express": "^4.17.17",
    "@types/jest": "^29.5.2",
    "@types/node": "^20.3.1",
    "@types/supertest": "^2.0.12",
    "@typescript-eslint/eslint-plugin": "^6.0.0",
    "@typescript-eslint/parser": "^6.0.0",
    "eslint": "^8.42.0",
    "eslint-config-prettier": "^9.0.0",
    "eslint-plugin-prettier": "^5.0.0",
    "jest": "^29.5.0",
    "prettier": "^3.0.0",
    "source-map-support": "^0.5.21",
    "supertest": "^6.3.3",
    "ts-jest": "^29.1.0",
    "ts-loader": "^9.4.3",
    "ts-node": "^10.9.1",
    "tsconfig-paths": "^4.2.0",
    "typescript": "^5.1.3"
  },
  "jest": {
    "moduleFileExtensions": [
      "js",
      "json",
      "ts"
    ],
    "rootDir": "src",
    "testRegex": ".*\\.spec\\.ts$",
    "transform": {
      "^.+\\.(t|j)s$": "ts-jest"
    },
    "collectCoverageFrom": [
      "**/*.(t|j)s"
    ],
```

```
    "coverageDirectory": "../coverage",
    "testEnvironment": "node"
  }
}
```

- **Rutas y CORS**
- En el main escribo app.enableCors() donde puedo configurar el whitelist, blacklist
- Genero el recurso gpt con nest g gpt --> elijo REST API
- Creo el endpoints en el controller

```
import { Body, Controller, Post } from '@nestjs/common';
import { GptService } from './gpt.service';
import { OrthographyDto } from './dtos';

@Controller('gpt')
export class GptController {

  constructor(private readonly gptService: GptService) {}


  @Post('orthography-check')
  orthographyCheck(
    @Body() orthographyDto: OrthographyDto,  //uso @Body para tomar lo que recibo
del body
  ) {
    return this.gptService.orthographyCheck(orthographyDto);
  }
}
```

- Mi servicio solo va a llamar a los casos de uso

```
import { Injectable } from '@nestjs/common';

import OpenAI from 'openai';

import { orthographyCheckUseCase } from './use-cases';
import { OrthographyDto } from './dtos';

@Injectable()
export class GptService {

  private openai = new OpenAI({
    apiKey: process.env.OPENAI_API_KEY, //le paswo el APIKEY
  })


  // Solo va a llamar casos de uso

  async orthographyCheck(orthographyDto: OrthographyDto) {
```

```
                                        //le paso la instancia de openai al caso
de uso
    return await orthographyCheckUseCase( this.openai, {
      prompt: orthographyDto.prompt // en el prompt le paso el contenido del body
    });
  }


}
```

- Para la variable de entorno instalo @nestjs/config y configuro el forRoot en app.module

```
import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';

import { GptModule } from './gpt/gpt.module';


@Module({
  imports: [
    ConfigModule.forRoot(),
    GptModule,
  ]
})
export class AppModule {}
```

- Coloco la API_KEY_OPENAI en el archivo .env en la raíz
- Este es el dto que le estoy pasando al caso de uso que valida el body desde el controller

```
import { IsInt, IsOptional, IsString } from 'class-validator';



export class OrthographyDto {

  @IsString()
  readonly prompt: string //readonly porque no lo voy a modificar

  @IsInt()
  @IsOptional()
  readonly maxTokens?: number; //maxTokens


}
```

- Configuro el class-vaslidator en el main con globalpipes

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { ValidationPipe } from '@nestjs/common';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.useGlobalPipes(
    new ValidationPipe({
      whitelist: true,
      forbidNonWhitelisted: true,
    }),
  );

  app.enableCors();

  await app.listen(3000);
}
bootstrap();
```

- El caso de uso (creo la carpeta use-cases dentro de src/gpt/)

```
import OpenAI from 'openai';

//creo la interfaz de options
interface Options {
  prompt: string;
}


export const orthographyCheckUseCase = async( openai: OpenAI,  options: Options )
=> {

  const { prompt } = options;

  //envío el prompt a openAI
  const completion = await openai.chat.completions.create({
    messages: [
      {
        role: "system", // el role de opeanAI, assistant es para desarrolladores,
tengo tool, function, user
        //ern content me devolverá la respuesta
        content: `
        Te serán proveídos textos en español con posibles errores ortográficos y
gramaticales,
        Las palabras usadas deben de existir en el diccionario de la Real Academia
Española,
        Debes de responder en formato JSON,
        tu tarea es corregirlos y retornar información soluciones,
        también debes de dar un porcentaje de acierto por el usuario,
```

```
        Si no hay errores, debes de retornar un mensaje de felicitaciones.

        Ejemplo de salida:
        {
          userScore: number,
          errors: string[], // ['error -> solución']
          message: string, //  Usa emojis y texto para felicitar al usuario
        }


        `
      },
      //en un segundo objeto le paSO EL ROLE Y EL CONTENT
      {
        role: 'user', //role de usuario
        content: prompt, //le paso el prompt
      }
  ],
    //le paso el modelo y la config al objeto principal
    model: "gpt-4o",

    temperature: 0.3, //de 0 a 2, cuuanto valores más altos las respuestas serán
más aleatorias
    max_tokens: 150, //número máximo de tokens que puede usar para la completación
    response_format: {
      type: 'json_object' //devuelvo la respuesta como un json, no todos
losmodelos soportan este formato
    }
  });

  // console.log(completion);
  const jsonResp = JSON.parse(completion.choices[0].message.content); //parseo la
respuesta para retornarla
                              //En el arreglo de choices, está en el content,
dentro de messages

  return jsonResp;

}
```

- Más adelante veremos como podemos crear un thread para poder pasarle info a openai varias veces en un mismo contexto en diferentes consultas
- También se puede mostrar la respuesta que va generando con los streams de información

---

Consumo del caso de uso en el frontend

- Creo en src/core/use-cases/ortography.use-case.ts

```
export const ortographyUseCase =async (prompt: string)=>{
    try {
                                    //la url del meu backend
`http://localhost:3000/gpt/orthography-check`
        const resp = await fetch(`${import.meta.env.VITE_GPOT_API}/orthography-
check`, {
            //como no es un get, es un POST, debo indicarlo
            method: 'POST',
            headers:{
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({prompt}) //en el body de la petición le paso el
prompt
        })

        if(!resp.ok) throw new Error ("No se pudo realizar la conexión")

            const data = await resp.json() //de esta manera la respuesta es de
tipo any
            //hago una petición y uso paste code as JSON para sacar la interfaz


    } catch (error) {
        //tiparemos esta respuesta
        return{
            ok: false,
            userScore: 0,
            errors: [],
            mnessage: "NO SE PUDO REALIZAR LA CORRECCIÓN"
        }
    }
}
```

- Para tipar la respuesta de la petición fetch, hago una petición y uso pastre code as JSON y saco la interfaz
- La guardo en interfaces/orthography.response.ts

```
export interface OrthographgyResponse{
    ok?: boolean
    useScore: number
    errors: []
    message: string

}
```

- Sigo con el caso de uso

```
import { OrthographgyResponse } from "../../interfaces/orthography.response"

export const ortographyUseCase =async (prompt: string):
Promise<OrthographgyResponse>=>{
    try {
                                    //la url del meu backend
`http://localhost:3000/gpt/orthography-check`
        const resp = await fetch(`${import.meta.env.VITE_GPOT_API}/orthography-
check`, {
            //como no es un get, es un POST, debo indicarlo
            method: 'POST',
            headers:{
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({prompt}) //en el body de la petición le paso el
prompt
        })

        if(!resp.ok) throw new Error ("No se pudo realizar la conexión")

            const data= await resp.json() as OrthographgyResponse

            return{
                ok: true,
                ...data
            }

    } catch (error) {
        //tiparemos esta respuesta
        return{
            ok: false,
            useScore: 0,
            errors: [],
            message: "NO SE PUDO REALIZAR LA CORRECCIÓN"
        }
    }
}
```

- Exporto el caso de uso en el index de la carpèta use-cases

```
export * from './orthography.use-case'
```

- Creo el archivo .env con parte de la url para comunicarme con el backend

```
VITE_GPT_API =http://localhost:3000/gpt
```

- En src/presentation/pages/orthography/OrthographyPage.ts

```
import { useState } from 'react';
import { GptMessage, MyMessage, TextMessageBox, TypingLoader } from
"../../components";
import { ortographyUseCase } from '../../../core/use-cases';

interface Message {
  text: string;
  isGpt: boolean;
}




export const OrthographyPage = () => {

  const [isLoading, setIsLoading] = useState(false);
  const [messages, setMessages] = useState<Message[]>([])


  const handlePost = async( text: string ) => {

    setIsLoading(true);
    setMessages( (prev) => [...prev, { text: text, isGpt: false }] );

    //UseCase
    const data = await ortographyUseCase(text)

    setIsLoading(false);

    // Todo: Añadir el mensaje de isGPT en true


  }



  return (
    <div className="chat-container">
      <div className="chat-messages">
        <div className="grid grid-cols-12 gap-y-2">
          {/* Bienvenida */}
          <GptMessage text="Hola, puedes escribir tu texto en español, y te ayudo
con las correcciones" />

          {
            messages.map( (message, index) => (
              message.isGpt
                ? (
                  <GptMessage key={ index } text="Esto es de OpenAI" />
                )
                : (
                  <MyMessage key={ index } text={ message.text } />
                )
```

```
            ))
          }


          {
            isLoading && (
              <div className="col-start-1 col-end-12 fade-in">
                <TypingLoader />
              </div>
            )
          }


        </div>
      </div>


      <TextMessageBox
        onSendMessage={ handlePost }
        placeholder='Escribe aquí lo que deseas'
        disableCorrections
      />

    </div>
  );
};
```

Mostrar en pantalla la info

- Añado info como opcional a la interfaz de Message (estoy en OrthographyPage)
- Envio los mensajes con setMessage, evalúo si viene unan respuesta, le paso la data

```
import { useState } from 'react';
import { GptMessage, MyMessage, TextMessageBox, TypingLoader } from
"../../components";
import { ortographyUseCase } from '../../../core/use-cases';

interface Message {
  text: string;
  isGpt: boolean;
  info?:{
    useScore: number,
    errors: string[],
    message: string
  }
}

export const OrthographyPage = () => {

  const [isLoading, setIsLoading] = useState(false);
```

```
  const [messages, setMessages] = useState<Message[]>([])


  const handlePost = async( text: string ) => {

    setIsLoading(true);
    setMessages( (prev) => [...prev, { text: text, isGpt: false }] );

    //UseCase
    const data = await ortographyUseCase(text) //puedo desestructurar de aquí el
ok, useScore, message y errors
    //evaluamos si el ok está en true
    if(!data.ok){    //prev es para no perder los messages anteriores
      setMessages( (prev) => [...prev, { text: "No se pudo realizar la
corrección", isGpt: false }] );

    }else{
      setMessages( (prev) => [...prev,
      { text:data.message,
        isGpt: false,
        info:{
          errors: data.errors,
          useScore: data.useScore,
          message: data.message
        }
      }]);
    }

    setIsLoading(false);

    // Todo: Añadir el mensaje de isGPT en true


  }



  return (
    <div className="chat-container">
      <div className="chat-messages">
        <div className="grid grid-cols-12 gap-y-2">
          {/* Bienvenida */}
          <GptMessage text="Hola, puedes escribir tu texto en español, y te ayudo
con las correcciones" />

          {
            messages.map( (message, index) => (
              message.isGpt
                ? (
                  <GptMessage key={ index } text="Esto es de OpenAI" /> //me
devuelve ESTO porque tengo la data en duro
                )
                : (
                  <MyMessage key={ index } text={ message.text } />
```

```
                )

            ))
          }


          {
            isLoading && (
              <div className="col-start-1 col-end-12 fade-in">
                <TypingLoader />
              </div>
            )
          }


        </div>
      </div>


      <TextMessageBox
        onSendMessage={ handlePost }
        placeholder='Escribe aquí lo que deseas'
        disableCorrections
      />

    </div>
  );
};
```

- En poantalla me devuelve "Esto es de OpenAI" por que lo estoy mandando en duro en el messages.map
- Creemos un GptMessage que se encargue de recibir el objeto tal cual quiuero usarlo
- Hago una copia del GptMessage y lo llamo presentation/components/chat-bubbles/GptMessageOrthography

```
import Markdown from "react-markdown";

interface Props {
  userScore: number
  errors: string[]
  message: string
}

export const GptOrthographyMessage = ({ userScore, message, errors}: Props) => {
  return (
    <div className="col-start-1 col-end-9 p-3 rounded-lg">
      <div className="flex flex-row items-start">
        <div className="flex items-center justify-center h-10 w-10 rounded-full
bg-green-600 flex-shrink-0">
          G
        </div>
```

```
        <div className="relative ml-3 text-sm bg-black bg-opacity-25 pt-3 pb-2 px-
4 shadow rounded-xl">
         <h3 className="text-3xl">Puntaje : {userScore}</h3>
         <p>{message}</p>
         {
          (errors.length === 0)
          ? <p>No se encontraron errores</p>
          : (
            <>
            <h3 className="text-2xl">Errores encontrados</h3>
            <ul>
              {
                errors.map((error, i)=>(
                  <li key={i}>
                      {error}
                  </li>
                ))
              }
            </ul>
            </>
          )
         }
        </div>
      </div>
    </div>
  );
};
```

- Lo uso para renderizar en OrthograpohyPage
- No es message.info?, nos hemos asegurado de que siempre lo vamos a tener, por eso lo marcamos como message.info!
- Puedo usar un spread del message.info!

```
 return (
    <div className="chat-container">
      <div className="chat-messages">
        <div className="grid grid-cols-12 gap-y-2">
          {/* Bienvenida */}
          <GptMessage text="Hola, puedes escribir tu texto en español, y te ayudo
con las correcciones" />

          {
            messages.map( (message, index) => (
              message.isGpt
                ? (
                <GptOrthographyMessage key={index}
                  errors= {message.info!.errors}
                  userScore={message.info!.userScore}
                  message={message.info!.message}
                  //o puedo poner solo {...message.info!}
                />
```

```
                )
                : (
                  <MyMessage key={ index } text={ message.text } />
                )

            ))
          }


          {
            isLoading && (
              <div className="col-start-1 col-end-12 fade-in">
                <TypingLoader />
              </div>
            )
          }


        </div>
      </div>


      <TextMessageBox
        onSendMessage={ handlePost }
        placeholder='Escribe aquí lo que deseas'
        disableCorrections
      />

    </div>
  );
```

02 OpenAI Backend- ProsCons Discusser - Streams (backend)

ProsCons Discusser - controllers, service y use-case

- ProsCons Discusser es un asistente que analiza los pros y los contras de algo
- gpt.controller

```
import { Body, Controller, HttpStatus, Post, Res } from '@nestjs/common';
import { Response } from 'express';

import { GptService } from './gpt.service';
import { OrthographyDto, ProsConsDiscusserDto } from './dtos';

@Controller('gpt')
export class GptController {

  constructor(private readonly gptService: GptService) {}
```

```
  @Post('orthography-check')
  orthographyCheck(
    @Body() orthographyDto: OrthographyDto,
  ) {
    return this.gptService.orthographyCheck(orthographyDto);
  }


  @Post('pros-cons-discusser')
  prosConsDicusser(
    @Body() prosConsDiscusserDto: ProsConsDiscusserDto,
  ) {
    return this.gptService.prosConsDicusser(prosConsDiscusserDto);
  }

  @Post('pros-cons-discusser-stream')
  async prosConsDicusserStream(
    @Body() prosConsDiscusserDto: ProsConsDiscusserDto,
    @Res() res: Response, //cuando capto la response con @Res en NEST debo crear
la respuesta que voy a emitir. Un return no funcionará
  ) {
     const stream = await
this.gptService.prosConsDicusserStream(prosConsDiscusserDto); //obtengo el stream

    //creo la respuesta que voy a emitir
    res.setHeader('Content-Type', 'application/json'); //seteo los headers, voy a
regresar un json
    res.status( HttpStatus.OK ); //Status nativo de NEST

    //uso el awaait en el for para recorrer los chunks de manera asíncrona
    for await( const chunk of stream ) {
      const piece = chunk.choices[0].delta.content || '';
      // console.log(piece); van apareciendo fragmentos de la respuesta en consola
      res.write(piece); //escribo en la response cada chunk de la respuesta de
OpenAI
    }

    res.end(); //cierro la conexión

  }


}
```

- ProsConsDiscusserDto

```
import { IsString } from 'class-validator';

export class ProsConsDiscusserDto {
```

```
  @IsString()
  readonly prompt: string;

}
```

- gpt.service

```
import { Injectable } from '@nestjs/common';

import OpenAI from 'openai';

import { orthographyCheckUseCase, prosConsDicusserStreamUseCase,
prosConsDicusserUseCase } from './use-cases';
import { OrthographyDto, ProsConsDiscusserDto } from './dtos';

@Injectable()
export class GptService {

  private openai = new OpenAI({
    apiKey: process.env.OPENAI_API_KEY,
  })


  // Solo va a llamar casos de uso

  async orthographyCheck(orthographyDto: OrthographyDto) {
    return await orthographyCheckUseCase( this.openai, {
      prompt: orthographyDto.prompt
    });
  }

  async prosConsDicusser({ prompt }: ProsConsDiscusserDto ) {
    return await prosConsDicusserUseCase(this.openai, { prompt });
  }

  async prosConsDicusserStream({ prompt }: ProsConsDiscusserDto ) {
    return await prosConsDicusserStreamUseCase(this.openai, { prompt });
  }



}
```

- pros-cons-discusser.use-case

```
import OpenAI from 'openai';

interface Options {
  prompt: string;
```

```
}


//le indico que me devuelva la respuesta en formato markdown en el prompt
export const prosConsDicusserUseCase = async (openai: OpenAI, { prompt }: Options)
=> {

  const response = await openai.chat.completions.create({
    model: 'gpt-4',
    messages: [
      {
        role: 'system',
        content: `
          Se te dará una pregunta y tu tarea es dar una respuesta con pros y
contras,
          la respuesta debe de ser en formato markdown,
          los pros y contras deben de estar en una lista,
        `
      },
      {
        role: 'user',
        content: prompt
      }
    ],
    temperature: 0.8, //0.8 es un poco aleatoria
    max_tokens: 500 //puede ser que la respuesta salga cortada porque limito
tokens para una respuesta que puede ser extensa
  })



  return response.choices[0].message; //puedo poner .content para que lo que
devuelva sea un string en lugar de un JSON
        //response.choices[0].message.content --> devuelve un string
}
```

- ProsConsDiscusserStreamUseCase
- Simplemente coloco **el stream en true**

```
import OpenAI from 'openai';

interface Options {
  prompt: string;
}



export const prosConsDicusserStreamUseCase = async (openai: OpenAI, { prompt }:
Options) => {

  //retorno el stream
```

```
  return await openai.chat.completions.create({
    stream: true,
    model: 'gpt-4',
    messages: [
      {
        role: 'system',
        content: `
          Se te dará una pregunta y tu tarea es dar una respuesta con pros y
contras,
          la respuesta debe de ser en formato markdown,
          los pros y contras deben de estar en una lista,
        `
      },
      {
        role: 'user',
        content: prompt
      }
    ],
    temperature: 0.8,
    max_tokens: 500
  })

}
```

OpenAI Nest + React - Streams (frontend)

- El caso de uso de pros y cons (sin el stream) es muy similar al de orthography
- Hago un try catch
- Uso fetch con el método POST
- Le paso la url del endpoint
- Configuro fetch con method, los headers
- Le paso el body usando JSOPN.stringify
- Si no hay respuesta.ok lanzo un error
- Si la hay, guardo resp.json en data como ProsConsResponse
- Retorno un objeto con el ok en true y esparzo la data
- En el catch recojo el error
- Retorno un objeto con el ok en false y en content el string de información del error
- En src/core/use-cases/pros-cons.use-case

```
import type { ProsConsResponse } from '../../interfaces';



export const prosConsUseCase = async( prompt: string ) => {

  try {

    const resp = await fetch(`${ import.meta.env.VITE_GPT_API }/pros-cons-
```

```
discusser`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ prompt })
    });

    if ( !resp.ok ) throw new Error('No se pudo realizar la comparación');

    const data = await resp.json() as ProsConsResponse;


    return  {
      ok: true,
      ...data,
    }


  } catch (error) {
    return {
      ok: false,
      content: 'No se pudo realizar la comparación'
    }
  }


}
```

- Uso paste JSON as code para sacar la interfaz de la respuesta de OpenAI
- src/interfaces/proscons...

```
// Generated by https://quicktype.io

export interface ProsConsResponse {
  role:    string;
  content: string;
}
```

- En presentation/pages/ProsConsPage

```
import { useState } from 'react';
import { GptMessage, MyMessage, TextMessageBox, TypingLoader } from
'../../components';
import { prosConsUseCase } from '../../../core/use-cases';


interface Message {
  text: string;
```

```
  isGpt: boolean;
}




export const ProsConsPage = () => {

  const [isLoading, setIsLoading] = useState(false);
  const [messages, setMessages] = useState<Message[]>([])


  const handlePost = async( text: string ) => {

    setIsLoading(true);
    setMessages( (prev) => [...prev, { text: text, isGpt: false }] );


    const { ok, content } = await prosConsUseCase( text );
    setIsLoading(false);

    if ( !ok ) return;


    setMessages( (prev) => [...prev, { text: content, isGpt: true }] );


  }



  return (
    <div className="chat-container">
      <div className="chat-messages">
        <div className="grid grid-cols-12 gap-y-2">
          {/* Bienvenida */}
          <GptMessage text="Puedes escribir lo que sea que quieres que compare y
te de mis puntos de vista." />

          {
            messages.map( (message, index) => (
              message.isGpt
                ? (
                  <GptMessage key={ index } text={ message.text } />
                )
                : (
                  <MyMessage key={ index } text={ message.text } />
                )

            ))
          }


          {
```

```
            isLoading && (
              <div className="col-start-1 col-end-12 fade-in">
                <TypingLoader />
              </div>
            )
          }


        </div>
      </div>


      <TextMessageBox
        onSendMessage={ handlePost }
        placeholder='Escribe aquí lo que deseas'
        disableCorrections
      />

    </div>
  );
};
```

- Paso los componentes GptMessage, MyMessage, TYpingLoader¡ y TextMessageBox
- src/components/chat-bubble/GptMessage

```
import Markdown from "react-markdown";

interface Props {
  text: string;
}

export const GptMessage = ({ text }: Props) => {
  return (
    <div className="col-start-1 col-end-9 p-3 rounded-lg">
      <div className="flex flex-row items-start">
        <div className="flex items-center justify-center h-10 w-10 rounded-full
bg-green-600 flex-shrink-0">
          G
        </div>
        <div className="relative ml-3 text-sm bg-black bg-opacity-25 pt-3 pb-2 px-
4 shadow rounded-xl">
          <Markdown>{text}</Markdown>
        </div>
      </div>
    </div>
  );
};
```

- MyMessage

```
interface Props {
  text: string;
}

export const MyMessage = ({ text }: Props) => {
  return (
    <div className="col-start-6 col-end-13 p-3 rounded-lg">
      <div className="flex items-center justify-start flex-row-reverse">
        <div className="flex items-center justify-center h-10 w-10 rounded-full
bg-indigo-500 flex-shrink-0">
          F
        </div>
        <div className="relative mr-3 text-sm bg-indigo-700 py-2 px-4 shadow
rounded-xl">
          <div>{ text }</div>
        </div>
      </div>
    </div>
  );
};
```

- src/components/chat-input/TextMessageBox

```
import { FormEvent, useState } from 'react';


interface Props {
  onSendMessage: (message: string)=>void;
  placeholder?: string;
  disableCorrections?: boolean;
}


export const TextMessageBox = ({ onSendMessage, placeholder, disableCorrections =
false }: Props) => {

  const [message, setMessage] = useState('')



  const handleSendMessage = (event: FormEvent<HTMLFormElement>) => {
    event.preventDefault();

    if ( message.trim().length === 0 ) return;

    onSendMessage( message );
    setMessage('');
  }

  return (
    <form
```

```
      onSubmit={ handleSendMessage }
      className="flex flex-row items-center h-16 rounded-xl bg-white w-full px-4"
    >

      <div className="flex-grow">
        <div className="relative w-full">

          <input
            type="text"
            autoFocus
            name="message"
            className="flex w-full border rounded-xl text-gray-800 focus:outline-
none focus:border-indigo-300 pl-4 h-10"
            placeholder={ placeholder }
            autoComplete={ disableCorrections ? 'on': 'off' }
            autoCorrect={ disableCorrections ? 'on': 'off' }
            spellCheck={ disableCorrections ? 'true': 'false' }
            value={ message }
            onChange={ (e) => setMessage( e.target.value ) }
          />


        </div>
      </div>


      <div className="ml-4">
          <button className="btn-primary">
            <span className="mr-2">Enviar</span>
            <i className="fa-regular fa-paper-plane"></i>
          </button>
      </div>




    </form>
  )
}
```

- src/components/loaders/TypingLoader

```
.typing {
  display: block;
  width: 60px;
  height: 40px;
  border-radius: 20px;
  margin: 0 1rem;
  display: flex;
  justify-content: center;
  align-items: center;
```

```
  background-color: #f2f2f2;
}

.circle {
  display: block;
  height: 10px;
  width: 10px;
  border-radius: 50%;
  background-color: #8d8d8d;
  margin: 3px;
}
.circle.scaling {
  animation: typing 1000ms ease-in-out infinite;
  animation-delay: 3600ms;
}
.circle.bouncing {
  animation: bounce 1000ms ease-in-out infinite;
  animation-delay: 3600ms;
}

.circle:nth-child(1) {
  animation-delay: 0ms;
}

.circle:nth-child(2) {
  animation-delay: 333ms;
}

.circle:nth-child(3) {
  animation-delay: 666ms;
}

@keyframes typing {
  0% {
    transform: scale(1);
  }
  33% {
    transform: scale(1);
  }
  50% {
    transform: scale(1.4);
  }
  100% {
    transform: scale(1);
  }
}

@keyframes bounce {
  0% {
    transform: translateY(0);
  }
  33% {
    transform: translateY(0);
  }
```

```
  50% {
    transform: translateY(-10px);
  }
  100% {
    transform: translateY(0);
  }
}
```

- El caso de uso de pros-cons-stream.use-case copio el chat-template y le cambio el nombre
- Hago el fetch en un try catch
- Creo el reader
- Creo un decoder para mostrar los mensajes porque la info va a venir poco a poco
- TextDecoder viene en JS
- Creo la variable let text donde iré concatenando lo que el stream me vaya proporcionando
- Del reader con read extraigo el value y done
- Cuando tenga el done es que ha acabado la transmisión, cierro con un break el ciclo while
- En decodedCHunk guardo con decoder.decode el value, debo decirle que esto viene como un stream
- Concateno en text el decodedChunk

```
export const prosConsStreamUseCase = async( prompt: string ) => {

  try {

    const resp = await fetch(`${ import.meta.env.VITE_GPT_API }/pros-cons-
discusser-stream`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ prompt }),
      // todo: abortSignal
    });

    if ( !resp.ok ) throw new Error('No se pudo realizar la comparación');

    //genero el reader
    const reader = resp.body?.getReader();
    if ( !reader ) {
      console.log('No se pudo generar el reader');
      return null;
    }

    return reader;

    // const decoder = new TextDecoder();

    // let text = '';

    // while( true ) {
    //   const { value, done } = await reader.read();
```

```
    //   if ( done ) {
    //     break;
    //   }

    //   const decodedChunk = decoder.decode( value, { stream: true } );
    //   text += decodedChunk;
    //   console.log(text);

    // }




  } catch (error) {
    console.log(error);
    return null;
  }


}
```

- Ahora ya tengo la respuesta en consola. debo guardarla en un estado para poder mostrarla en pantalla
- Vamos a ProsCOnsStreamPage
- Comento el decoder en el use-case, lo haré en otro lugar. retorno el reader
- Obtengo el reader haciendo uso del caso de uso
- Lo que quiero es ir mostrando el mensaje en pantall mientras va siendo emitido
- También vamos a programar que si se escribe algo mientras se está retronando una respuesta aborte con AbortSignal
- Una vez obtengo el reader pongo el isLoading en false
- Creo el decoder y el message
- Genero el nuevo mensaje con setMessages con isGpt en true
- Uso un while para controlar el stream y decodificar los chunk
- Tenemos que actualizar el último mensaje (el creado con setMessage), no tenemos que crear uno nuevo
- Actualizo el último mensaje (se refactorizará)

```
import { useRef, useState } from 'react';
import { GptMessage, MyMessage, TypingLoader, TextMessageBox } from
'../../components';
import { prosConsStreamGeneratorUseCase } from '../../../core/use-cases';

interface Message {
  text: string;
  isGpt: boolean;
}


export const ProsConsStreamPage = () => {
```

```
  const [isLoading, setIsLoading] = useState(false);
  const [messages, setMessages] = useState<Message[]>([])


  const handlePost = async( text: string ) => {


    setIsLoading(true);
    setMessages( (prev) => [...prev, { text: text, isGpt: false }] );

    //Aqui voy a tener el reader
    const reader = prosConsStreamGeneratorUseCase(text);
    setIsLoading(false);//una vez obtengo el reader pongo el isLoading en false



    const decoder = new TextDecoder()
    let message = ''

    //genero un nuevo mensaje
    setMessages( (messages) => [ ...messages, { text: '', isGpt: true  } ] );

    while(true){
        const {value, done} = await reader.read()
        if(done){
            break;
        }

        const decodedChunk = decoder.decode(value, {stream: true})
        message+=decodedChunk
    }

        //actualizar el útimo mensaje
        setMessages( (messages) => {
        const newMessages = [...messages]; //esparzo los messages anteriores
        newMessages[ newMessages.length - 1 ].text = message; //length -1 para
obtener el último mensaje
                                                //le digo que el texto del último
mensaje será igual al mensaje que estoy generando
        return newMessages;
      });

  }



  return (
    <div className="chat-container">
      <div className="chat-messages">
        <div className="grid grid-cols-12 gap-y-2">
          {/* Bienvenida */}
          <GptMessage text="¿Qué deseas comparar hoy?" />

          {
```

```
            messages.map( (message, index) => (
              message.isGpt
                ? (
                  <GptMessage key={ index } text={ message.text } />
                )
                : (
                  <MyMessage key={ index } text={ message.text } />
                )

            ))
          }


          {
            isLoading && (
              <div className="col-start-1 col-end-12 fade-in">
                <TypingLoader />
              </div>
            )
          }


        </div>
      </div>


      <TextMessageBox
        onSendMessage={ handlePost }
        placeholder='Escribe aquí lo que deseas'
        disableCorrections
      />

    </div>
  );
};
```

- El componente tiene mucha lógica relacionada a la construcción del mensaje

---

Stream con función generadora

- Hago uso del yield para retornar un valor sin cortar el flujo
- Copio el use-case de stream y añado el asterisco a function y en lugar de usar un return uso yield
- pros-cons-stream-generator.use-case

```
//las funciones generadoras llevan un asterisco al final de function
export async function* prosConsStreamGeneratorUseCase( prompt: string)  {

  try {

    const resp = await fetch(`${ import.meta.env.VITE_GPT_API }/pros-cons-
```

```
discusser-stream`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ prompt }),
    });

    if ( !resp.ok ) throw new Error('No se pudo realizar la comparación');

    const reader = resp.body?.getReader();
    if ( !reader ) {
      console.log('No se pudo generar el reader');
      return null;
    }


    const decoder = new TextDecoder();

    let text = '';

    while( true ) {
      const { value, done } = await reader.read();
      if ( done ) {
        break;
      }

      const decodedChunk = decoder.decode( value, { stream: true } );
      text += decodedChunk;
      // console.log(text);
      yield text; //regreso el texto
    }


  } catch (error) {
    console.log(error);
    return null;
  }


}
```

- En ProsConsStreamPage ya no necesito el reader, ni el while, etc...
- Llamo al caso de uso y pongo el Loading en false
- Creo el mensaje donde voy a estar haciendo el append de la información
- Uso el for await con el mismo código del setMessages, solo que lo guardo en la variable texto del for
- **Uso de AbortSignals para cancelar el stream**
    - Para cancelar la información generada si mando otra consulta
    - AbortController ya viene en JS
    - Necesito pasárselo al objeto que hace la emisión
    - Uso abortControler.current y uso .signal

- Le paso al caso de uso ProsConsStreamGeneratorUseCase el abortSignal

```
import { useRef, useState } from 'react';
import { GptMessage, MyMessage, TypingLoader, TextMessageBox } from
'../../components';
import { prosConsStreamGeneratorUseCase } from '../../../core/use-cases';

interface Message {
  text: string;
  isGpt: boolean;
}




export const ProsConsStreamPage = () => {

  const abortController = useRef( new AbortController() );
  const isRunning = useRef(false)

  const [isLoading, setIsLoading] = useState(false);
  const [messages, setMessages] = useState<Message[]>([])


  const handlePost = async( text: string ) => {

    if ( isRunning.current ) {
      abortController.current.abort(); //para abortar
      abortController.current = new AbortController();//creo una nueva señal
    }



    setIsLoading(true);
    isRunning.current = true; //en este punto pongo el isRunning en true
    setMessages( (prev) => [...prev, { text: text, isGpt: false }] );

    //TODO: UseCase
    const stream = prosConsStreamGeneratorUseCase( text,
abortController.current.signal ); //le paso el signal para que cuando reciba esta
señal cancele
    setIsLoading(false);


    setMessages( (messages) => [ ...messages, { text: '', isGpt: true  } ] );

    for await (const text of stream) {
      setMessages( (messages) => {
        const newMessages = [...messages];
        newMessages[ newMessages.length - 1 ].text = text;
        return newMessages;
      });
    }
```

```
    isRunning.current = false; //acabado el trabajo lo pongo a false

  }



  return (
    <div className="chat-container">
      <div className="chat-messages">
        <div className="grid grid-cols-12 gap-y-2">
          {/* Bienvenida */}
          <GptMessage text="¿Qué deseas comparar hoy?" />

          {
            messages.map( (message, index) => (
              message.isGpt
                ? (
                  <GptMessage key={ index } text={ message.text } />
                )
                : (
                  <MyMessage key={ index } text={ message.text } />
                )

            ))
          }


          {
            isLoading && (
              <div className="col-start-1 col-end-12 fade-in">
                <TypingLoader />
              </div>
            )
          }


        </div>
      </div>


      <TextMessageBox
        onSendMessage={ handlePost }
        placeholder='Escribe aquí lo que deseas'
        disableCorrections
      />

    </div>
  );
};
```

- ProsConsStreamGeneratorUseCase

- Necesito pasarle al fetch el abortSignal

```
export async function* prosConsStreamGeneratorUseCase( prompt: string,
abortSignal: AbortSignal )  {

  try {

    const resp = await fetch(`${ import.meta.env.VITE_GPT_API }/pros-cons-
discusser-stream`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ prompt }),
      signal: abortSignal, //le paso el abortSignal
    });

    if ( !resp.ok ) throw new Error('No se pudo realizar la comparación');

    const reader = resp.body?.getReader();
    if ( !reader ) {
      console.log('No se pudo generar el reader');
      return null;
    }


    const decoder = new TextDecoder();

    let text = '';

    while( true ) {
      const { value, done } = await reader.read();
      if ( done ) {
        break;
      }

      const decodedChunk = decoder.decode( value, { stream: true } );
      text += decodedChunk;
      // console.log(text);
      yield text;
    }


  } catch (error) {
    console.log(error);
    return null;
  }


}
```

OpenAI Nest + React - Traduccio0n (backend y frontend)

- gpt.controller

```
import { Body, Controller, HttpStatus, Post, Res } from '@nestjs/common';
import { Response } from 'express';

import { GptService } from './gpt.service';
import { OrthographyDto, ProsConsDiscusserDto, TranslateDto } from './dtos';

@Controller('gpt')
export class GptController {

  constructor(private readonly gptService: GptService) {}


  @Post('orthography-check')
  orthographyCheck(
    @Body() orthographyDto: OrthographyDto,
  ) {
    return this.gptService.orthographyCheck(orthographyDto);
  }


  @Post('pros-cons-discusser')
  prosConsDicusser(
    @Body() prosConsDiscusserDto: ProsConsDiscusserDto,
  ) {
    return this.gptService.prosConsDicusser(prosConsDiscusserDto);
  }

  @Post('pros-cons-discusser-stream')
  async prosConsDicusserStream(
    @Body() prosConsDiscusserDto: ProsConsDiscusserDto,
    @Res() res: Response,
  ) {
     const stream = await
this.gptService.prosConsDicusserStream(prosConsDiscusserDto);


    res.setHeader('Content-Type', 'application/json');
    res.status( HttpStatus.OK );

    for await( const chunk of stream ) {
      const piece = chunk.choices[0].delta.content || '';
      // console.log(piece);
      res.write(piece);
    }

    res.end();
```

```
  }

  //endpoint translate
  @Post('translate')
  translateText(
    @Body() translateDto: TranslateDto,
  ) {
    return this.gptService.translateText(translateDto);
  }


}
```

- El translateDto

```
import { IsString } from 'class-validator';

export class TranslateDto {
  @IsString()
  readonly prompt: string;

  @IsString()
  readonly lang: string;
}
```

- En el gpt.service

```
 async translateText({ prompt, lang }: TranslateDto ) {
    return await translateUseCase(this.openai, { prompt, lang });
  }
```

- El caso de uso

```
import OpenAI from 'openai';

interface Options {
  prompt: string;
  lang: string;
}



export const translateUseCase = async (openai: OpenAI, { prompt, lang }: Options)
=> {

  const response = await openai.chat.completions.create({
```

```
    model: 'gpt-4',
    messages: [
      {
        role: 'system',
        content: `Traduce el siguiente texto al idioma ${lang}:${ prompt }`
      },
    ],
    temperature: 0.2,
    // max_tokens: 500
  })



  return { message: response.choices[0].message.content }
}
```

- Para el frontend copio el chatTemplate
- presentation/pages/translate/TranslatePage

```
import { useState } from "react";
import { GptMessage, MyMessage, TypingLoader, TextMessageBoxSelect } from
'../../components';
import { translateTextUseCase } from '../../../core/use-cases';

interface Message {
  text: string;
  isGpt: boolean;
}

const languages = [
  { id: "alemán", text: "Alemán" },
  { id: "árabe", text: "Árabe" },
  { id: "bengalí", text: "Bengalí" },
  { id: "francés", text: "Francés" },
  { id: "hindi", text: "Hindi" },
  { id: "inglés", text: "Inglés" },
  { id: "japonés", text: "Japonés" },
  { id: "mandarín", text: "Mandarín" },
  { id: "portugués", text: "Portugués" },
  { id: "ruso", text: "Ruso" },
];

export const TranslatePage = () => {
  const [isLoading, setIsLoading] = useState(false);
  const [messages, setMessages] = useState<Message[]>([]);

  const handlePost = async (text: string, selectedOption: string) => {
    setIsLoading(true);

    const newMessage = `Traduce: "${ text }" al idioma ${ selectedOption }`
    setMessages((prev) => [...prev, { text: newMessage, isGpt: false }]);
```

```
    const { ok, message } = await translateTextUseCase( text, selectedOption )
    setIsLoading(false);
    if ( !ok ) { //si no tengo el ok, lanzo una alerta
      return alert(message);
    }

    setMessages((prev) => [...prev, { text: message, isGpt: true }]); //ai tengo
el ok le paso todos los mensajes anteriores con el spread y mando el message al
state
  };

  return (
    <div className="chat-container">
      <div className="chat-messages">
        <div className="grid grid-cols-12 gap-y-2">
          {/* Bienvenida */}
          <GptMessage text="¿Qué quieres que traduzca hoy?" />

          {messages.map((message, index) =>
            message.isGpt ? (
              <GptMessage key={index} text={ message.text } />
            ) : (
              <MyMessage key={index} text={message.text} />
            )
          )}

          {isLoading && (
            <div className="col-start-1 col-end-12 fade-in">
              <TypingLoader />
            </div>
          )}
        </div>
      </div>

      <TextMessageBoxSelect
        onSendMessage={handlePost}
        placeholder="Escribe aquí lo que deseas"
        options={ languages }
      />
    </div>
  );
};
```

- components/chat-bubbles/GPTMessage

```
import Markdown from "react-markdown";

interface Props {
  text: string;
}
```

```
export const GptMessage = ({ text }: Props) => {
  return (
    <div className="col-start-1 col-end-9 p-3 rounded-lg">
      <div className="flex flex-row items-start">
        <div className="flex items-center justify-center h-10 w-10 rounded-full
bg-green-600 flex-shrink-0">
          G
        </div>
        <div className="relative ml-3 text-sm bg-black bg-opacity-25 pt-3 pb-2 px-
4 shadow rounded-xl">
          <Markdown>{text}</Markdown>
        </div>
      </div>
    </div>
  );
};
```

- MyMessage

```
interface Props {
  text: string;
}

export const MyMessage = ({ text }: Props) => {
  return (
    <div className="col-start-6 col-end-13 p-3 rounded-lg">
      <div className="flex items-center justify-start flex-row-reverse">
        <div className="flex items-center justify-center h-10 w-10 rounded-full
bg-indigo-500 flex-shrink-0">
          F
        </div>
        <div className="relative mr-3 text-sm bg-indigo-700 py-2 px-4 shadow
rounded-xl">
          <div>{ text }</div>
        </div>
      </div>
    </div>
  );
};
```

OpenAI Nest + React (backend) - Texto a Audio

- Aqui almacenaremos el archivo de audio en fileSystem pero en la vida real se haría en la nube
- controller.ts

```
 @Get('text-to-audio/:fileId')
  async textToAudioGetter(
    @Res() res: Response,
```

```
    @Param('fileId') fileId: string,
  ) {
    const filePath = await this.gptService.textToAudioGetter(fileId);

    res.setHeader('Content-Type','audio/mp3');
    res.status(HttpStatus.OK);
    res.sendFile(filePath);

  }


  @Post('text-to-audio')
  async textToAudio(
    @Body() textToAudioDto: TextToAudioDto,
    @Res() res: Response,
  ) {
    const filePath = await this.gptService.textToAudio(textToAudioDto);

    res.setHeader('Content-Type','audio/mp3');
    res.status(HttpStatus.OK);
    res.sendFile(filePath);

  }
```

- El textToAudioDto

```
import { IsOptional, IsString } from 'class-validator';

export class TextToAudioDto {

  @IsString()
  readonly prompt: string;

  @IsString()
  @IsOptional()
  readonly voice?: string; //modelo de voz a elegir
}
```

- En el service. hay que pasarlo a Buffer para poder escribirlo en sistema

```
async textToAudio({ prompt, voice }: TextToAudioDto) {
    return await textToAudioUseCase(this.openai, { prompt, voice });
  }

 //le paso el id (el nombre que generé al mp3)
 async textToAudioGetter(fileId: string) {

   //resuelvo el path con un template string pasándole el nombre
   const filePath = path.resolve(
```

```
      __dirname,
      '../../generated/audios/',
      `${fileId}.mp3`,
    );

    //busco en el fileSystem que esté el archivo
    const wasFound = fs.existsSync(filePath);

  //si no está mando una excepción
    if (!wasFound) throw new NotFoundException(`File ${fileId} not found`);

  //si está lo retorno
    return filePath;
  }
```

- El useCase

```
import * as path from 'path';
import * as fs from 'fs';

import OpenAI from 'openai';

interface Options {
  prompt: string;
  voice?: string; //voice opcional
}

//le paso el openAI y las options
export const textToAudioUseCase = async (
  openai: OpenAI,
  { prompt, voice }: Options,
) => {

  //tipos de voz
  const voices = {
    nova: 'nova',
    alloy: 'alloy',
    echo: 'echo',
    fable: 'fable',
    onyx: 'onyx',
    shimmer: 'shimmer',
  };

    //si no viene una voice en el body de la request, por defecto será nova
  const selectedVoice = voices[voice] ?? 'nova';

//indico el path donde se guardarán los archivos
  const folderPath = path.resolve(__dirname, '../../../generated/audios/');
  //creo el nombre que será la data en formato número.mp3
  const speechFile = path.resolve(`${folderPath}/${new Date().getTime()}.mp3`);

    //creo el directorio con la ruta
```

```
  fs.mkdirSync(folderPath, { recursive: true });

    //creo el mp3 usando OpneAI
  const mp3 = await openai.audio.speech.create({
    model: 'tts-1',
    voice: selectedVoice, //le paso la voz
    input: prompt, //le paso el prompt
    response_format: 'mp3',
  });

    //tengo que pasarlo a Buffer para escribirlo en sistema
  const buffer = Buffer.from( await mp3.arrayBuffer() );
  fs.writeFileSync( speechFile, buffer );//le paso el nombre del archivo que he
creado y el buffer


  return speechFile; //retorno la ruta con el nombre del archivo
};
```

Frontend

- Vamos a llamar al endpoint desde el frontend
- Está regresando un mp3, recibo un bloque de info (no un json)
- Debo colocarlo en un elemento de audio html para reproducirlo

```
export const textToAudioUseCase = async (prompt: string, voice: string) => {
  try {
    const resp = await fetch(`${import.meta.env.VITE_GPT_API}/text-to-audio`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ prompt, voice }),
    });

    if (!resp.ok)
      throw new Error("No se pudo realizar la generación del audio");

    const audioFile = await resp.blob(); //uso .blob porque retorna un mp3 (no
tengo resp.json)
    const audioUrl = URL.createObjectURL(audioFile); //genero un URL que se pueda
colocar en un audioTypeElement

    console.log({audioUrl});

    return { ok: true, message: prompt, audioUrl: audioUrl };

  } catch (error) {
```

```
    return {
      ok: false,
      message: "No se pudo realizar la generación del audio",
    };
  }
};
```

- TextToAudioPage

```
import { useState } from "react";
import {
  GptMessage,
  MyMessage,
  TypingLoader,
  TextMessageBox,
  TextMessageBoxSelect,
  GptMessageAudio,
} from "../../components";
import { textToAudioUseCase } from "../../../core/use-cases";

const displaimer = `## ¿Qué audio quieres generar hoy?
* Todo el audio generado es por AI.
`;

const voices = [
  { id: "nova", text: "Nova" },
  { id: "alloy", text: "Alloy" },
  { id: "echo", text: "Echo" },
  { id: "fable", text: "Fable" },
  { id: "onyx", text: "Onyx" },
  { id: "shimmer", text: "Shimmer" },
];

interface TextMessage {
  text: string;
  isGpt: boolean;
  type: "text";
}

interface AudioMessage {
  text: string;
  isGpt: boolean;
  audio: string;
  type: "audio";
}

type Message = TextMessage | AudioMessage;

export const TextToAudioPage = () => {
  const [isLoading, setIsLoading] = useState(false);
  const [messages, setMessages] = useState<Message[]>([]);
```

```
  const handlePost = async (text: string, selectedVoice: string) => {
    setIsLoading(true);
    setMessages((prev) => [
      ...prev,
      { text: text, isGpt: false, type: "text" },
    ]);

    //TODO: UseCase
    const { ok, message, audioUrl } = await textToAudioUseCase(
      text,
      selectedVoice
    );
    setIsLoading(false);

    if (!ok) return;

    setMessages((prev) => [
      ...prev,
      {
        text: `${selectedVoice} - ${message}`,
        isGpt: true,
        type: "audio",
        audio: audioUrl!,
      },
    ]);
  };

  return (
    <div className="chat-container">
      <div className="chat-messages">
        <div className="grid grid-cols-12 gap-y-2">
          {/* Bienvenida */}
          <GptMessage text={displaimer} />

          {messages.map((message, index) =>
            message.isGpt ? (
              message.type === "audio" ? (
                <GptMessageAudio
                  key={index}
                  text={message.text}
                  audio={message.audio}
                />
              ) : (
                <GptMessage key={index} text={message.text} />
              )
            ) : (
              <MyMessage key={index} text={message.text} />
            )
          )}

          {isLoading && (
            <div className="col-start-1 col-end-12 fade-in">
              <TypingLoader />
            </div>
```

```
          )}
        </div>
      </div>

      <TextMessageBoxSelect
        onSendMessage={handlePost}
        placeholder="Escribe aquí lo que deseas"
        options={voices}
      />
    </div>
  );
};
```

- GptMessageAudio

```
import Markdown from "react-markdown";

interface Props {
  text: string;
  audio: string;
}

export const GptMessageAudio = ({ text, audio }: Props) => {
  return (
    <div className="col-start-1 col-end-9 p-3 rounded-lg">
      <div className="flex flex-row items-start">
        <div className="flex items-center justify-center h-10 w-10 rounded-full
bg-green-600 flex-shrink-0">
          G
        </div>
        <div className="relative ml-3 text-sm bg-black bg-opacity-25 pt-3 pb-2 px-
4 shadow rounded-xl">
          <Markdown>{text}</Markdown>
          <audio
            controls
            src={ audio }
            className="w-full"
            autoPlay
          />
        </div>
      </div>
    </div>
  );
};
```

OpenAI Nest + React - Audio a texto

- *NOTA*: en la sección de frontend están el resto de componentes de la aplicación no mostrados en la documentación anterior

- AudiToTextUseCase

```
import * as fs from 'fs';

import OpenAI from 'openai';

interface Options {
  prompt?: string; //el prompt será opcional
  audioFile: Express.Multer.File;
}


export const audioToTextUseCase = async( openai:OpenAI, options: Options ) => {

  const { prompt, audioFile} = options;

  // console.log({ prompt, audioFile });

  const response = await openai.audio.transcriptions.create({
    model: 'whisper-1',
    file: fs.createReadStream( audioFile.path ),
    prompt: prompt, // mismo idioma del audio
    language: 'es',//DEBE SEGUIR EL ISO6391. Debe de estar en el mismo idioma que
el audio
    // response_format: 'vtt', // 'srt',
    response_format: 'verbose_json',
  })


  return response;

}
```

- En el controller
- Uso un audio de menos de 25 MB
- diskStorage viene de multer (npm i multer)
- Debo pasarlke el prompt y el file en POSTMAN

```
  @Post('audio-to-text')
  @UseInterceptors(
    FileInterceptor('file', { //intercepto el file
      storage: diskStorage({ //uso diskStorage para indicar la ubicación de
destino
        destination: './generated/uploads',
        filename: (req, file, callback) => {
          const fileExtension = file.originalname.split('.').pop(); //extraigo la
extensión
```

```
          const fileName = `${ new Date().getTime() }.${ fileExtension }`;//creoi
el nombre del archivo
          return callback(null, fileName);
        }
      })//no estamos validando que sea una extensión permitida, lo validamos en el
método de abajo
    })
  )
  async audioToText(
    @UploadedFile(//uso este decorador para trabajar con el file subido
      new ParseFilePipe({ //hago la validación
        validators: [
          new MaxFileSizeValidator({ maxSize: 1000 * 1024 * 5, message: 'File is
bigger than 5 mb ' }),//defino el tamaño máximo (5MB)
          new FileTypeValidator({ fileType: 'audio/*' })//que sea cualquier tipo
audio
        ]
      })
    ) file: Express.Multer.File,
  ) {

    return this.gptService.audioToText(file); //esto regresará algo de tipo
Edxpress.Multer.File
  }
```

- gpt.service

```
async audioToText( audioFile: Express.Multer.File, prompt?: string ) {
return await audioToTextUseCase( this.openai, { audioFile, prompt } );
}
```

- De esta manera regresará un archivo de texto a modo de subtitulos indicando de que segundo a que segundo con el audio transcrito
- Depende de la salida que le indique en response_format: 'srt', 'vtt',
- 'verbose_json' es útil porque da mucha más info (útil para el frontend)

Frontend

- src/core/use-cases

```
import type { AudioToTextResponse } from '../../interfaces';


export const audioToTextUseCase = async( audioFile: File, prompt?: string ) => {

  try {

    const formData = new FormData();
```

```
    formData.append('file', audioFile );
    if ( prompt ) {
      formData.append('prompt', prompt );
    }



    const resp = await fetch(`${ import.meta.env.VITE_GPT_API }/audio-to-text`, {
      method: 'POST',
      body: formData
    });

    const data = await resp.json() as AudioToTextResponse;
    return data;


  } catch (error) {
    console.log(error);
    return null;
  }


}
```

- src/interfaces

```
// Generated by https://quicktype.io

export interface AudioToTextResponse {
  task:     string;
  language: string;
  duration: number;
  text:     string;
  segments: Segment[];
}

export interface Segment {
  id:                number;
  seek:              number;
  start:             number;
  end:               number;
  text:              string;
  tokens:            number[];
  temperature:       number;
  avg_logprob:       number;
  compression_ratio: number;
  no_speech_prob:    number;
}
```

- presentation/pages

```
import { useState } from 'react';
import { GptMessage, MyMessage, TypingLoader, TextMessageBoxFile } from
'../../components';
import { audioToTextUseCase } from '../../../core/use-cases';

interface Message {
  text: string;
  isGpt: boolean;
}


export const AudioToTextPage = () => {

  const [isLoading, setIsLoading] = useState(false);
  const [messages, setMessages] = useState<Message[]>([])


  const handlePost = async( text: string, audioFile: File ) => {

    setIsLoading(true);
    setMessages( (prev) => [...prev, { text: text, isGpt: false }] );

    //TODO: UseCase
    const resp = await audioToTextUseCase(audioFile, text);
    setIsLoading(false);

    if ( !resp ) return; // no hay respuesta...

    const gptMessage = `
## Transcripción:
__Duración:__ ${ Math.round( resp.duration )  } segundos
## El texto es:
${ resp.text }
`


    setMessages( (prev) => [
      ...prev,
      { text: gptMessage, isGpt: true }
    ]);

    for( const segment of resp.segments ) {
      const segmentMessage = `
__De ${ Math.round( segment.start ) } a ${ Math.round( segment.end ) } segundos:__
${ segment.text }
`

      setMessages( (prev) => [
        ...prev,
        { text: segmentMessage, isGpt: true }
      ]);
    }
```

```
  }


  return (
    <div className="chat-container">
      <div className="chat-messages">
        <div className="grid grid-cols-12 gap-y-2">
          {/* Bienvenida */}
          <GptMessage text="Hola, ¿qué audio quieres generar hoy?" />

          {
            messages.map( (message, index) => (
              message.isGpt
                ? (
                  <GptMessage key={ index } text={ message.text } />
                )
                : (
                  <MyMessage key={ index } text={ (message.text ===
''')?'Transcribe el audio': message.text } />
                )

            ))
          }


          {
            isLoading && (
              <div className="col-start-1 col-end-12 fade-in">
                <TypingLoader />
              </div>
            )
          }


        </div>
      </div>


      <TextMessageBoxFile
        onSendMessage={ handlePost }
        placeholder='Escribe aquí lo que deseas'
        disableCorrections
        accept="audio/*"
      />

    </div>
  );
};
```

-presentation/components//chat-bubble/GptMessage

```
import Markdown from "react-markdown";

interface Props {
  text: string;
}

export const GptMessage = ({ text }: Props) => {
  return (
    <div className="col-start-1 col-end-9 p-3 rounded-lg">
      <div className="flex flex-row items-start">
        <div className="flex items-center justify-center h-10 w-10 rounded-full
bg-green-600 flex-shrink-0">
          G
        </div>
        <div className="relative ml-3 text-sm bg-black bg-opacity-25 pt-3 pb-2 px-
4 shadow rounded-xl">
          <Markdown>{text}</Markdown>
        </div>
      </div>
    </div>
  );
};
```

-presentation/components//chat-bubble/GptMessageAudio

```
import Markdown from "react-markdown";

interface Props {
  text: string;
  audio: string;
}

export const GptMessageAudio = ({ text, audio }: Props) => {
  return (
    <div className="col-start-1 col-end-9 p-3 rounded-lg">
      <div className="flex flex-row items-start">
        <div className="flex items-center justify-center h-10 w-10 rounded-full
bg-green-600 flex-shrink-0">
          G
        </div>
        <div className="relative ml-3 text-sm bg-black bg-opacity-25 pt-3 pb-2 px-
4 shadow rounded-xl">
          <Markdown>{text}</Markdown>
          <audio
            controls
            src={ audio }
            className="w-full"
```

```
            autoPlay
          />
        </div>
      </div>
    </div>
  );
};
```

-presentation/components//chat-bubble/GptOrthographyMessage

```
interface Props {
  userScore: number;
  errors: string[];
  message: string;
}

export const GptOrthographyMessage = ({ userScore, errors, message }: Props) => {
  return (
    <div className="col-start-1 col-end-9 p-3 rounded-lg">
      <div className="flex flex-row items-start">
        <div className="flex items-center justify-center h-10 w-10 rounded-full
bg-green-600 flex-shrink-0">
          G
        </div>
        <div className="relative ml-3 text-sm bg-black bg-opacity-25 pt-3 pb-2 px-
4 shadow rounded-xl">

          <h3 className="text-3xl">Puntaje: { userScore }%</h3>
          <p>{ message }</p>

          {
            (errors.length === 0)
            ? <p>No se encontraron errores, perfecto!</p>
            : (
              <>
                <h3 className="text-2xl">Errores encontrados</h3>
                <ul>
                  {
                    errors.map( (error, i) => (
                      <li key={ i }>
                        { error }
                      </li>
                    ))
                  }
                </ul>
              </>
            )


          }
```

```
        </div>
      </div>
    </div>
  );
};
```

-presentation/components//chat-bubble/MyMessage

```
interface Props {
  text: string;
}

export const MyMessage = ({ text }: Props) => {
  return (
    <div className="col-start-6 col-end-13 p-3 rounded-lg">
      <div className="flex items-center justify-start flex-row-reverse">
        <div className="flex items-center justify-center h-10 w-10 rounded-full
bg-indigo-500 flex-shrink-0">
          F
        </div>
        <div className="relative mr-3 text-sm bg-indigo-700 py-2 px-4 shadow
rounded-xl">
          <div>{ text }</div>
        </div>
      </div>
    </div>
  );
};
```

- components/chat-input/textMessageBox

```
import { FormEvent, useState } from 'react';


interface Props {
  onSendMessage: (message: string)=>void;
  placeholder?: string;
  disableCorrections?: boolean;
}


export const TextMessageBox = ({ onSendMessage, placeholder, disableCorrections =
false }: Props) => {

  const [message, setMessage] = useState('')
```

```
  const handleSendMessage = (event: FormEvent<HTMLFormElement>) => {
    event.preventDefault();

    if ( message.trim().length === 0 ) return;

    onSendMessage( message );
    setMessage('');
  }

  return (
    <form
      onSubmit={ handleSendMessage }
      className="flex flex-row items-center h-16 rounded-xl bg-white w-full px-4"
    >

      <div className="flex-grow">
        <div className="relative w-full">

          <input
            type="text"
            autoFocus
            name="message"
            className="flex w-full border rounded-xl text-gray-800 focus:outline-
none focus:border-indigo-300 pl-4 h-10"
            placeholder={ placeholder }
            autoComplete={ disableCorrections ? 'on': 'off' }
            autoCorrect={ disableCorrections ? 'on': 'off' }
            spellCheck={ disableCorrections ? 'true': 'false' }
            value={ message }
            onChange={ (e) => setMessage( e.target.value ) }
          />


        </div>
      </div>


      <div className="ml-4">
          <button className="btn-primary">
            <span className="mr-2">Enviar</span>
            <i className="fa-regular fa-paper-plane"></i>
          </button>
      </div>




    </form>
  )
}
```

- chat-input/textMessageboxFile

```tsx
import { FormEvent, useRef, useState } from 'react';


interface Props {
  onSendMessage: (message: string, file: File )=>void;
  placeholder?: string;
  disableCorrections?: boolean;
  accept?: string; // image/*
}


export const TextMessageBoxFile = ({ onSendMessage, placeholder,
disableCorrections = false, accept }: Props) => {

  const [message, setMessage] = useState('');

  const [selectedFile, setSelectedFile] = useState<File | null>()
  const inputFileRef = useRef<HTMLInputElement>(null);




  const handleSendMessage = (event: FormEvent<HTMLFormElement>) => {
    event.preventDefault();

    // if ( message.trim().length === 0 ) return;
    if ( !selectedFile ) return;

    onSendMessage( message, selectedFile );
    setMessage('');
    setSelectedFile(null);
  }

  return (
    <form
      onSubmit={ handleSendMessage }
      className="flex flex-row items-center h-16 rounded-xl bg-white w-full px-4"
    >
      <div className="mr-3">
        <button
          type="button"
          className="flex items-center justify-center text-gray-400 hover:text-
gray-600"
          onClick={ () => inputFileRef.current?.click() }
        >
            <i className="fa-solid fa-paperclip text-xl"></i>
        </button>

        <input
          type="file"
          ref={ inputFileRef }
          accept={ accept }
```

```
          onChange={ (e) => setSelectedFile( e.target.files?.item(0) ) }
          hidden
        />

        </div>



        <div className="flex-grow">
          <div className="relative w-full">

            <input
              type="text"
              autoFocus
              name="message"
              className="flex w-full border rounded-xl text-gray-800 focus:outline-
none focus:border-indigo-300 pl-4 h-10"
              placeholder={ placeholder }
              autoComplete={ disableCorrections ? 'on': 'off' }
              autoCorrect={ disableCorrections ? 'on': 'off' }
              spellCheck={ disableCorrections ? 'true': 'false' }
              value={ message }
              onChange={ (e) => setMessage( e.target.value ) }
            />


          </div>
        </div>


        <div className="ml-4">
            <button
              className="btn-primary"
              disabled={ !selectedFile }
            >
              {
                ( !selectedFile )
                 ? <span className="mr-2">Enviar</span>
                 : <span className="mr-2"> { selectedFile.name.substring(0,10) +
'...' }  </span>
              }
              <i className="fa-regular fa-paper-plane"></i>
            </button>
        </div>




      </form>
    )
}
```

- TextMessageBoxSelect

```
import { FormEvent, useState } from 'react';


interface Props {
  onSendMessage: (message: string, selectedOption: string )=>void;
  placeholder?: string;
  disableCorrections?: boolean;
  options: Option[];
}

interface Option {
  id: string;
  text: string;
}



export const TextMessageBoxSelect = ({ onSendMessage, placeholder,
disableCorrections = false, options }: Props) => {

  const [message, setMessage] = useState('');
  const [selectedOption, setSelectedOption] = useState<string>('');



  const handleSendMessage = (event: FormEvent<HTMLFormElement>) => {
    event.preventDefault();

    if ( message.trim().length === 0 ) return;
    if ( selectedOption === '' ) return;

    onSendMessage( message, selectedOption );
    setMessage('');
  }

  return (
    <form
      onSubmit={ handleSendMessage }
      className="flex flex-row items-center h-16 rounded-xl bg-white w-full px-4"
    >

      <div className="flex-grow">
        <div className="flex">

          <input
            type="text"
            autoFocus
            name="message"
            className="w-full border rounded-xl text-gray-800 focus:outline-none
focus:border-indigo-300 pl-4 h-10"
            placeholder={ placeholder }
            autoComplete={ disableCorrections ? 'on': 'off' }
```

```
            autoCorrect={ disableCorrections ? 'on': 'off' }
            spellCheck={ disableCorrections ? 'true': 'false' }
            value={ message }
            onChange={ (e) => setMessage( e.target.value ) }
          />

          <select
            name="select"
            className="w-2/5 ml-5 border rounded-xl text-gray-800 focus:outline-
none focus:border-indigo-300 pl-4 h-10"
            value={ selectedOption }
            onChange={ e => setSelectedOption( e.target.value ) }
          >
            <option value=''>Seleccione</option>
            {
              options.map( ({ id, text }) => (
                <option key={ id } value={ id }>{ text }</option>
              ))
            }
          </select>


        </div>
      </div>


      <div className="ml-4">
          <button className="btn-primary">
            <span className="mr-2">Enviar</span>
            <i className="fa-regular fa-paper-plane"></i>
          </button>
      </div>




    </form>
  )
}
```

- components/loaders/TypingLoader

```
.typing {
  display: block;
  width: 60px;
  height: 40px;
  border-radius: 20px;
  margin: 0 1rem;
  display: flex;
  justify-content: center;
  align-items: center;
  background-color: #f2f2f2;
```

```
}

.circle {
  display: block;
  height: 10px;
  width: 10px;
  border-radius: 50%;
  background-color: #8d8d8d;
  margin: 3px;
}
.circle.scaling {
  animation: typing 1000ms ease-in-out infinite;
  animation-delay: 3600ms;
}
.circle.bouncing {
  animation: bounce 1000ms ease-in-out infinite;
  animation-delay: 3600ms;
}

.circle:nth-child(1) {
  animation-delay: 0ms;
}

.circle:nth-child(2) {
  animation-delay: 333ms;
}

.circle:nth-child(3) {
  animation-delay: 666ms;
}

@keyframes typing {
  0% {
    transform: scale(1);
  }
  33% {
    transform: scale(1);
  }
  50% {
    transform: scale(1.4);
  }
  100% {
    transform: scale(1);
  }
}

@keyframes bounce {
  0% {
    transform: translateY(0);
  }
  33% {
    transform: translateY(0);
  }
  50% {
```

```
    transform: translateY(-10px);
  }
  100% {
    transform: translateY(0);
  }
}
```

- TypingLoader

```
import './TypingLoader.css';

interface Props {
  className?: string;
}


export const TypingLoader = ({ className }: Props) => {
  return (
    <div className={`typing ${ className }`}>
      <span className="circle scaling"></span>
      <span className="circle scaling"></span>
      <span className="circle scaling"></span>
    </div>
  )
}
```

- components/sidebar/SideBarMenuItem

```
import { NavLink } from 'react-router-dom';

interface Props {
  to: string;
  icon: string;
  title: string;
  description: string;
}


export const SidebarMenuItem = ({
  to, icon, title, description
}:Props) => {
  return (
    <NavLink
      to={to}
      className={({ isActive }) =>
        isActive
          ? "flex justify-center items-center bg-gray-800 rounded-md p-2
transition-colors"
          : "flex justify-center items-center hover:bg-gray-800 rounded-md p-2
```

```
transition-colors"
      }
    >
      <i className={`${icon} text-2xl mr-4 text-indigo-400`}></i>
      <div className="flex flex-col flex-grow">
        <span className="text-white text-lg font-semibold">{title}</span>
        <span className="text-gray-400 text-sm">{description}</span>
      </div>
    </NavLink>
  );
};
```

- presentation/layouts/Dashboard (layout)

```
import { NavLink, Outlet } from "react-router-dom";
import { menuRoutes } from '../router/router';
import { SidebarMenuItem } from '../components';

export const DashboardLayout = () => {
  return (
    <main className="flex flex-row mt-7">
      <nav className="hidden sm:flex flex-col ml-5 w-[370px] min-h-[calc(100vh-
3.0rem)] bg-white bg-opacity-10 p-5 rounded-3xl">
        <h1 className="font-bold text-lg lg:text-3xl bg-gradient-to-br from-white
via-white/50 bg-clip-text text-transparent">
          ReactGPT<span className="text-indigo-500">.</span>
        </h1>
        <span className="text-xl">Bienvenido</span>

        <div className="border-gray-700 border my-3" />

        {/* Opciones del menú */}
        {
          menuRoutes.map( option => (
            <SidebarMenuItem key={option.to} {...option } />
          ))
        }

      </nav>

      <section className="mx-3 sm:mx-20 flex flex-col w-full h-[calc(100vh-50px)]
bg-white bg-opacity-10 p-5 rounded-3xl">
        <div className="flex flex-row h-full">
          <div className="flex flex-col flex-auto h-full p-1">
            <Outlet />
          </div>
        </div>
      </section>
    </main>
  );
};
```

- presentation/router/router

```
import { Navigate, createBrowserRouter } from 'react-router-dom';
import { OrthographyPage, ProsConsPage, ProsConsStreamPage, TranslatePage,
TextToAudioPage, ImageGenerationPage, AssistantPage, ImageTunningPage,
AudioToTextPage } from '../pages';
import { DashboardLayout } from '../layouts/DashboardLayout';

export const menuRoutes = [
  {
    to: "/orthography",
    icon: "fa-solid fa-spell-check",
    title: "Ortografía",
    description: "Corregir ortografía",
    component: <OrthographyPage />
  },
  {
    to: "/pros-cons",
    icon: "fa-solid fa-code-compare",
    title: "Pros & Cons",
    description: "Comparar pros y contras",
    component: <ProsConsPage />
  },
  {
    to: "/pros-cons-stream",
    icon: "fa-solid fa-water",
    title: "Como stream",
    description: "Con stream de mensajes",
    component: <ProsConsStreamPage />
  },
  {
    to: "/translate",
    icon: "fa-solid fa-language",
    title: "Traducir",
    description: "Textos a otros idiomas",
    component: <TranslatePage />
  },
  {
    to: "/text-to-audio",
    icon: "fa-solid fa-podcast",
    title: "Texto a audio",
    description: "Convertir texto a audio",
    component: <TextToAudioPage />
  },
  {
    to: "/audio-to-text",
    icon: "fa-solid fa-comment-dots",
    title: "Audio a texto",
    description: "Convertir audio a texto",
    component: <AudioToTextPage />
  },
  {
```

```
    to: "/image-generation",
    icon: "fa-solid fa-image",
    title: "Imágenes",
    description: "Generar imágenes",
    component: <ImageGenerationPage />
  },
  {
    to: "/image-tunning",
    icon: "fa-solid fa-wand-magic",
    title: "Editar imagen",
    description: "Generación continua",
    component: <ImageTunningPage />
  },

  {
    to: "/assistant",
    icon: "fa-solid fa-user",
    title: "Asistente",
    description: "Información del asistente",
    component: <AssistantPage />
  },
];


export const router = createBrowserRouter([
  {
    path: "/",
    element: <DashboardLayout />,
    children: [
      ...menuRoutes.map( route => ({
        path: route.to,
        element: route.component
      })),
      {
        path: '',
        element: <Navigate to={ menuRoutes[0].to } />
      }

    ],
  }
])
```

- index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;

html, body {
  background-color: #000000;
  color: white;
  font-family: 'Open Sans', sans-serif;
```

```
}


h1 {
  @apply text-3xl font-bold mb-4;
}

p {
  @apply mb-4;
}

ul {
  @apply list-disc list-inside;
}

strong {
  @apply font-bold text-indigo-400 text-xl;
}

em {
  @apply italic text-pink-500;
}


.btn-primary {
  @apply bg-indigo-500 text-white font-bold py-2 px-4 rounded-xl hover:bg-indigo-
700 transition-all duration-200 ease-in-out;
}

.btn-primary:disabled {
  @apply bg-indigo-500 text-white font-bold py-2 px-4 rounded-xl opacity-50
cursor-not-allowed;
}

.chat-container {
  @apply flex flex-col flex-auto flex-shrink-0 rounded-2xl bg-white bg-opacity-5
h-full p-4;
}

.chat-messages {
  @apply flex flex-col h-full overflow-x-auto mb-4 overflow-scroll;
}


/* Animations */

.fade-in { animation: fadeIn .3s; }

@keyframes fadeIn {
  0% { opacity: 0; }
  100% { opacity: 1; }
}
```

- main

```
import React from 'react'
import ReactDOM from 'react-dom/client'

import { ReactGPT } from './ReactGPT';
import './index.css'

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <ReactGPT />
  </React.StrictMode>,
)
```

- ReactGPT

```
import { RouterProvider } from 'react-router-dom';
import { router } from './presentation/router/router';

export const ReactGPT = () => {
  return (
    <RouterProvider router={ router } />
  )
}
```

- index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>ReactGPT</title>

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.5.1/css/all.min.css" integrity="sha512-
DTOQO9RWCH3ppGqcWaEA1BIZOC6xxalwEsw9c2QQeAIftl+Vegovlnee1c9QX4TctnWMn13TZye+giMm8e
2LwA==" crossorigin="anonymous" referrerpolicy="no-referrer" />
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

OpenAI Nest + react - Generación de imágenes

- En el controller

```
@Post('image-generation')
  async imageGeneration(@Body() imageGenerationDto: ImageGenerationDto) {
    return await this.gptService.imageGeneration(imageGenerationDto);
  }

  @Get('image-generation/:filename')
  async getGenerated(@Res() res: Response, @Param('filename') fileName: string) {
    const filePath = this.gptService.getGeneratedImage(fileName);
    res.status(HttpStatus.OK);
    res.sendFile(filePath);
  }

   @Post('image-variation')
  async imageVariation(@Body() imageVariationDto: ImageVariationDto) {
    return await this.gptService.geneateImageVariation(imageVariationDto);
  }
```

- Dto
- image-generation

```
import { IsOptional, IsString } from 'class-validator';


export class ImageGenerationDto {

  @IsString()
  readonly prompt: string;


  @IsString()
  @IsOptional()
  readonly originalImage?: string;

  @IsString()
  @IsOptional()
  readonly maskImage?: string;


}
```

- image-variation

```
import { IsString } from 'class-validator';



export class ImageVariationDto {

  @IsString()
  readonly baseImage: string;

}
```

- use-case/image-generation
- Necesito poder mandar el originalImage y el maskImage desde el frontend para poder generar la edición de la foto
- Para decirle que trabaje sobre, por ejemplo, cambiarle un ojo al personaje generado, le quitaré el ojo con GIMP (o Adobe) y dejaré visible la máscara que hay detrás
- Debemos mandarle el png con esa transparencia (canal ALPHA)
- Lo hago en el método download image as png
- Entonces si viene una imagen y una máscara lo que quiere es la edición
- La edición solo funcioan con Dall-e-2 por el momento
- Como digo, necesito borrar el contorno y mandarle las dos imágenes (la original y la imagen con la zona recortada dónde quiero la variación)
- Si no recibo la maskImage ni la originalImage, creo una imagen nueva con el prompt (que si es obligatorio desde la interface)
-

```
import * as fs from 'fs';
import * as path from 'path';

import OpenAI from 'openai';
import { downloadBase64ImageAsPng, downloadImageAsPng } from 'src/helpers';

interface Options {
  prompt: string;
  originalImage?: string;
  maskImage?: string;
}

export const imageGenerationUseCase = async (
  openai: OpenAI,
  options: Options,
) => {
  const { prompt, originalImage, maskImage } = options;

  // Si el originalImage o el maskInage no vienen ejecuto el generate
  if (!originalImage || !maskImage) {
    const response = await openai.images.generate({
      prompt: prompt,
      model: 'dall-e-3',
```

```
      n: 1, //número de imágenes, dall-e-3 solo soporta 1
      size: '1024x1024',
      quality: 'standard',
      response_format: 'url',
    });

    // Guardo en el FS
    const fileName = await downloadImageAsPng(response.data[0].url); //el url que
me devuele OPneAi de la imagen generada
    const url = `${process.env.SERVER_URL}/gpt/image-generation/${fileName}`;

    return {
      url: url,
      openAIUrl: response.data[0].url,
      revised_prompt: response.data[0].revised_prompt,
    };
  }

    //Si llego hasta aqui es que si tengo la originalImage y la maskImage
  // originalImage=http://localhost:3000/gpt/image-generation/1703770602518.png
   //la imagen en formato base64 será algo así
  // maskImage=Base64;ASDKJhaskljdasdlfkjhasdkjlHLKJDASKLJdashlkdjAHSKLJDhALSKJD

  const pngImagePath = await downloadImageAsPng(originalImage, true ); //obtengo
la imagen de mi backend
  const maskPath = await downloadBase64ImageAsPng(maskImage, true ); //obtengo la
máscara de mi backend

  const response = await openai.images.edit({
    model: 'dall-e-2',
    prompt: prompt,
    image: fs.createReadStream(pngImagePath), //le paso la imagen original
    mask: fs.createReadStream(maskPath), //la máscara
    n: 1, //solo devuelve una imagen
    size: '1024x1024',
    response_format: 'url',
  });

  const fileName = await downloadImageAsPng(response.data[0].url);
  const url = `${process.env.SERVER_URL}/gpt/image-generation/${fileName}`;

  return {
    url: url,
    openAIUrl: response.data[0].url,
    revised_prompt: response.data[0].revised_prompt,
  };
};
```

- use-cases/image-variation

```
import * as fs from 'fs';
```

```
import OpenAI from 'openai';
import { downloadImageAsPng } from 'src/helpers';

interface Options {
  baseImage: string;
}

export const imageVariationUseCase = async (
  openai: OpenAI,
  options: Options,
) => {
  const { baseImage } = options; //baseImage viene a

  const pngImagePath = await downloadImageAsPng( baseImage, true ); //obtengo la
imagen de mi backend

  // const response = await openai.images.createVariation({
  //   model: 'dall-e-2',
  //   image: fs.createReadStream(pngImagePath),
  //   n: 1,
  //   size: '1024x1024',
  //   response_format: 'url'
  // });
  const response = await openai.images.createVariation({
    model: 'dall-e-2',
    image: fs.createReadStream(pngImagePath),
    n: 1,
    size: '1024x1024',
    response_format: 'url'
  });

  const fileName = await downloadImageAsPng( response.data[0].url );
  const url = `${ process.env.SERVER_URL }/gpt/image-generation/${ fileName }`


  return {
    url: url,
    openAIUrl: response.data[0].url,
    revised_prompt: response.data[0].revised_prompt,
  }
};
```

- helpers/download-image-as-png
- instalo sharp con npm i sharp

```
import * as path from 'path';
import * as fs from 'fs';
import * as sharp from 'sharp';

import { InternalServerErrorException } from '@nestjs/common';
```

```
export const downloadImageAsPng = async (
  url: string,
  fullPath: boolean = false,
) => {
  const response = await fetch(url); //obtengo la imagen

  if (!response.ok) {
    throw new InternalServerErrorException('Download image was not possible');
  }

  const folderPath = path.resolve('./', './generated/images/');
  fs.mkdirSync(folderPath, { recursive: true });

  const imageNamePng = `${new Date().getTime()}.png`;
  const buffer = Buffer.from(await response.arrayBuffer());

  // fs.writeFileSync( `${ folderPath }/${ imageNamePng }`, buffer );
  const completePath = path.join(folderPath, imageNamePng);

  await sharp(buffer).png().ensureAlpha().toFile(completePath); //nos aseguramos
que sea un png y tenga el canal ALPHA!!

  return fullPath ? completePath : imageNamePng;
};

export const downloadBase64ImageAsPng = async (
  base64Image: string,
  fullPath: boolean = false,
) => {
  // Remover encabezado
  base64Image = base64Image.split(';base64,').pop();
  const imageBuffer = Buffer.from(base64Image, 'base64');

  const folderPath = path.resolve('./', './generated/images/');
  fs.mkdirSync(folderPath, { recursive: true });

  const imageNamePng = `${new Date().getTime()}-64.png`;

  const completePath = path.join(folderPath, imageNamePng);
  // Transformar a RGBA, png // Así lo espera OpenAI
  await sharp(imageBuffer).png().ensureAlpha().toFile(completePath);

  return fullPath ? completePath : imageNamePng;
};
```

- el servicio

```
 async imageGeneration( imageGenerationDto: ImageGenerationDto ) {
    return await imageGenerationUseCase( this.openai, { ...imageGenerationDto } );
  }

  getGeneratedImage( fileName: string ) {
```

```
    const filePath = path.resolve('./','./generated/images/', fileName);
    const exists = fs.existsSync( filePath );


    if ( !exists ) {
      throw new NotFoundException('File not found');
    }

    return filePath;
  }

  async geneateImageVariation( { baseImage }: ImageVariationDto ) {
    return imageVariationUseCase( this.openai, { baseImage } );
  }
```

Frontend

- image-generatrion-use-case

```
type GeneratedImage = Image | null;

interface Image {
  url: string;
  alt: string;
}

export const imageGenerationUseCase = async (
  prompt: string,
  originalImage?: string,
  maskImage?: string
): Promise<GeneratedImage> => {


  try {

    const resp = await fetch(`${ import.meta.env.VITE_GPT_API }/image-generation`,
{
      method: 'POST',
      headers: {
        'Content-Type':'application/json'
      },
      body: JSON.stringify({
        prompt,
        originalImage,
        maskImage,
      })
    });
```

```
    const {url,revised_prompt: alt } = await resp.json();

    return { url, alt };



  } catch (error) {
    console.log(error);
    return null;
  }



};
```

- image-variation-use-case

```
type GeneratedImage = Image | null;

interface Image {
  url: string;
  alt: string;
}

export const imageVariationUseCase = async (
  originalImage: string,
): Promise<GeneratedImage> => {


  try {

    const resp = await fetch(`${ import.meta.env.VITE_GPT_API }/image-variation`,
{
      method: 'POST',
      headers: {
        'Content-Type':'application/json'
      },
      body: JSON.stringify({
        baseImage: originalImage,
      })
    });


    const {url,revised_prompt: alt } = await resp.json();

    return { url, alt };



  } catch (error) {
    console.log(error);
```

```
    return null;
  }



};
```

- imageGenereationPage

```
import { useState } from "react";
import {
  GptMessage,
  MyMessage,
  TypingLoader,
  TextMessageBox,
  GptMessageImage,
} from "../../components";
import { imageGenerationUseCase } from '../../../core/use-cases';

interface Message {
  text: string;
  isGpt: boolean;
  info?: {
    imageUrl: string;
    alt: string;
  }
}

export const ImageGenerationPage = () => {
  const [isLoading, setIsLoading] = useState(false);
  const [messages, setMessages] = useState<Message[]>([]);

  const handlePost = async (text: string) => {
    setIsLoading(true);
    setMessages((prev) => [...prev, { text: text, isGpt: false }]);

    const imageInfo = await imageGenerationUseCase( text );
    setIsLoading(false);

    if ( !imageInfo ) {
      return setMessages( (prev) => [ ...prev, { text: 'No se pudo generar la
imagen', isGpt: true } ] );
    }

    setMessages( prev => [
      ...prev,
      {
        text: text,
        isGpt: true,
        info: {
          imageUrl: imageInfo.url,
          alt: imageInfo.alt
```

```
        }
      }
    ])


  };

  return (
    <div className="chat-container">
      <div className="chat-messages">
        <div className="grid grid-cols-12 gap-y-2">
          {/* Bienvenida */}
          <GptMessage text="¿Qué imagen deseas generar hoy?" />

          {messages.map((message, index) =>
            message.isGpt ? (
              <GptMessageImage
                key={index}
                text={ message.text }
                imageUrl={ message.info?.imageUrl! }
                alt={ message.info?.alt! }
              />
            ) : (
              <MyMessage key={index} text={message.text} />
            )
          )}

          {isLoading && (
            <div className="col-start-1 col-end-12 fade-in">
              <TypingLoader />
            </div>
          )}
        </div>
      </div>

      <TextMessageBox
        onSendMessage={handlePost}
        placeholder="Escribe aquí lo que deseas"
        disableCorrections
      />
    </div>
  );
};
```

- ImageTunningPage

```
import { useState } from "react";
import {
  GptMessage,
  MyMessage,
  TypingLoader,
  TextMessageBox,
```

```
  GptMessageImage,
  GptMessageSelectableImage,
} from "../../components";
import { imageGenerationUseCase, imageVariationUseCase } from "../../../core/use-
cases";

interface Message {
  text: string;
  isGpt: boolean;
  info?: {
    imageUrl: string;
    alt: string;
  };
}

export const ImageTunningPage = () => {
  const [isLoading, setIsLoading] = useState(false);
  const [messages, setMessages] = useState<Message[]>([
    {
      isGpt: true,
      text: 'Imagen base',
      info: {
        alt: 'Imagen base',
        imageUrl: 'http://localhost:3000/gpt/image-generation/1703785193790.png'
      }
    }
  ]);

  const [originalImageAndMask, setOriginalImageAndMask] = useState({
    original: undefined as
      | string
      | undefined,
    mask: undefined as string | undefined,
  });


  const handleVariation = async() => {
    setIsLoading(true);
    const resp = await imageVariationUseCase( originalImageAndMask.original! );
    setIsLoading(false);

    if ( !resp )return;

    setMessages( (prev) => [
      ...prev,
      {
        text: 'Variación',
        isGpt: true,
        info: {
          imageUrl: resp.url,
          alt: resp.alt
        }
      }
    ])
```

```
  }


  const handlePost = async (text: string) => {
    setIsLoading(true);
    setMessages((prev) => [...prev, { text: text, isGpt: false }]);

    const { original, mask } = originalImageAndMask;


    const imageInfo = await imageGenerationUseCase(text, original, mask );
    setIsLoading(false);

    if (!imageInfo) {
      return setMessages((prev) => [
        ...prev,
        { text: "No se pudo generar la imagen", isGpt: true },
      ]);
    }

    setMessages((prev) => [
      ...prev,
      {
        text: text,
        isGpt: true,
        info: {
          imageUrl: imageInfo.url,
          alt: imageInfo.alt,
        },
      },
    ]);
  };

  return (
    <>
      {
        originalImageAndMask.original && (
          <div className="fixed flex flex-col items-center top-10 right-10 z-10
fade-in">
            <span>Editando</span>
            <img
              className="border rounded-xl w-36 h-36 object-contain"
              src={ originalImageAndMask.mask ?? originalImageAndMask.original }
              alt="Imagen original"
            />
            <button onClick={ handleVariation } className="btn-primary mt-
2">Generar variación</button>
          </div>
        )
      }


      <div className="chat-container">
```

```
          <div className="chat-messages">
            <div className="grid grid-cols-12 gap-y-2">
              {/* Bienvenida */}
              <GptMessage text="¿Qué imagen deseas generar hoy?" />

              {messages.map((message, index) =>
                message.isGpt ? (
                  // <GptMessageImage
                  <GptMessageSelectableImage
                    key={index}
                    text={message.text}
                    imageUrl={message.info?.imageUrl!}
                    alt={message.info?.alt!}
                    onImageSelected={ (maskImageUrl) => setOriginalImageAndMask({
                      original: message.info?.imageUrl!,
                      mask: maskImageUrl
                    }) }
                  />
                ) : (
                  <MyMessage key={index} text={message.text} />
                )
              )}

              {isLoading && (
                <div className="col-start-1 col-end-12 fade-in">
                  <TypingLoader />
                </div>
              )}
            </div>
          </div>

          <TextMessageBox
            onSendMessage={handlePost}
            placeholder="Escribe aquí lo que deseas"
            disableCorrections
          />
        </div>
      </>
    );
};
```

backend ASSISTANT

- src/sam-assistant
- dtos/question.dto

```
import { IsString } from 'class-validator';
```

```
export class QuestionDto {

  @IsString()
  readonly threadId: string;

  @IsString()
  readonly question: string;

}
```

- sam assistant.controller

```
import { Body, Controller, Post } from '@nestjs/common';
import { SamAssistantService } from './sam-assistant.service';
import { QuestionDto } from './dtos/question.dto';

@Controller('sam-assistant')
export class SamAssistantController {

  constructor(private readonly samAssistantService: SamAssistantService) {}


  @Post('create-thread')
  async createThread() {
    return await this.samAssistantService.createThread();
  }

  @Post('user-question')
  async userQuestion(
    @Body() questionDto: QuestionDto
  ) {
    return await this.samAssistantService.userQuestion(questionDto);
  }
}
```

- sam-assistant.service

```
import { Injectable } from '@nestjs/common';

import OpenAI from 'openai';
import { checkCompleteStatusUseCase, createMessageUseCase, createRunUseCase,
createThreadUseCase, getMessageListUseCase } from './use-cases';
import { QuestionDto } from './dtos/question.dto';

@Injectable()
export class SamAssistantService {

  private openai = new OpenAI({
    apiKey: process.env.OPENAI_API_KEY,
```

```
  });


  async createThread() {
    return await createThreadUseCase( this.openai );
  }

  async userQuestion( questionDto: QuestionDto ) {
    const { threadId, question } = questionDto;

    const message = await createMessageUseCase(this.openai, { threadId, question
});

    const run = await createRunUseCase( this.openai, { threadId } );

    await checkCompleteStatusUseCase( this.openai, { runId: run.id, threadId:
threadId } );

    const messages = await getMessageListUseCase(this.openai, { threadId });

    return messages;
  }


}
```

- app.module

```
import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';

import { GptModule } from './gpt/gpt.module';
import { SamAssistantModule } from './sam-assistant/sam-assistant.module';


@Module({
  imports: [
    ConfigModule.forRoot(),
    GptModule,
    SamAssistantModule,
  ]
})
export class AppModule {}
```

-sam-assistant/ main

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { ValidationPipe } from '@nestjs/common';
```

```
import * as bodyParser from 'body-parser';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.useGlobalPipes(
    new ValidationPipe({
      whitelist: true,
      forbidNonWhitelisted: true,
    }),
  );

  app.enableCors();


  app.use( bodyParser.json({ limit: '10mb' }) );
  app.use( bodyParser.urlencoded({ limit: '10mb', extended: true }) );



  await app.listen(3000);
}
bootstrap();
```

- use-cases/
- check-complete-status-use-case

```
import OpenAI from 'openai';


interface Options {
  threadId: string;
  runId: string;
}


export const checkCompleteStatusUseCase = async( openai: OpenAI, options: Options
) => {

  const { threadId, runId } = options;

  const runStatus = await openai.beta.threads.runs.retrieve(
    threadId,
    runId
  );

  console.log({ status: runStatus.status}); // completed

  if ( runStatus.status === 'completed' ) {
    return runStatus;
```

```
  }

  // Esperar un segundo
  await new Promise( resolve => setTimeout( resolve, 1000 ) );

  return await checkCompleteStatusUseCase( openai, options );

}
```

- create-message-use-case

```
import OpenAI from 'openai';


interface Options {
  threadId: string;
  question: string;
}


export const createMessageUseCase = async ( openai: OpenAI, options: Options ) =>
{

  const { threadId, question } = options;

  const message = await openai.beta.threads.messages.create( threadId, {
    role: 'user',
    content: question,
  });

  return message;

}
```

- create-run-use-case

```
import OpenAI from 'openai';


interface Options {
  threadId: string;
  assistantId?: string;
}



export const createRunUseCase = async( openai: OpenAI, options: Options ) => {
```

```
  const { threadId, assistantId = 'asst_VVefvFU2YvJkLGhP4Yo521EN' } = options;

  const run = await openai.beta.threads.runs.create( threadId, {
    assistant_id: assistantId,
    // instructions; // OJO! Sobre escribe el asistente
  });

  console.log({run});

  return run;

}
```

- create-thread-use-case

```
import OpenAI from 'openai';

export const createThreadUseCase = async (openai: OpenAI) => {
  const { id } = await openai.beta.threads.create();
  return { id };
};
```

- get-message-list-use-case

```
import OpenAI from 'openai';


interface Options {
  threadId: string;
}


export const getMessageListUseCase = async( openai: OpenAI, options: Options ) =>
{

  const { threadId } = options;

  const messageList = await openai.beta.threads.messages.list( threadId );

  console.log( messageList );

  const messages = messageList.data.map( message => ({
    role: message.role,
    content: message.content.map( content => (content as any).text.value )
  }));

  return messages.reverse();
```

```
}
```

Frontend

- AssistantPage

```
import { useEffect, useState } from 'react';
import { GptMessage, MyMessage, TypingLoader, TextMessageBox } from
'../../components';
import { createThreadUseCase, postQuestionUseCase } from '../../../core/use-
cases';

interface Message {
  text: string;
  isGpt: boolean;
}



export const AssistantPage = () => {

  const [isLoading, setIsLoading] = useState(false);
  const [messages, setMessages] = useState<Message[]>([]);

  const [threadId, setThreadId] = useState<string>();

  // Obtener el thread, y si no existe, crearlo
  useEffect(() => {
    const threadId = localStorage.getItem('threadId');
    if ( threadId ) {
      setThreadId( threadId );
    } else {
      createThreadUseCase()
        .then( (id) => {
          setThreadId(id);
          localStorage.setItem('threadId', id)
        })
    }
  }, []);


  // useEffect(() => {
  //   if ( threadId ) {
  //     setMessages( (prev) => [ ...prev, { text: `Número de thread ${ threadId
}`, isGpt: true }] )
  //   }
  // }, [threadId])
```

```
  const handlePost = async( text: string ) => {

    if ( !threadId ) return;

    setIsLoading(true);
    setMessages( (prev) => [...prev, { text: text, isGpt: false }] );


    const replies = await postQuestionUseCase(threadId, text)

    setIsLoading(false);

    for (const reply of replies) {
      for (const message of reply.content) {
        setMessages ( (prev) => [
          ...prev,
          { text: message, isGpt: (reply.role === 'assistant'), info: reply  }
        ] )
      }
    }



  }



  return (
    <div className="chat-container">
      <div className="chat-messages">
        <div className="grid grid-cols-12 gap-y-2">
          {/* Bienvenida */}
          <GptMessage text="Buen día, soy Sam,¿Cuál es tu nombre? y ¿en qué puedo
ayudarte?" />

          {
            messages.map( (message, index) => (
              message.isGpt
                ? (
                  <GptMessage key={ index } text={ message.text } />
                )
                : (
                  <MyMessage key={ index } text={ message.text } />
                )

            ))
          }
```

```
          {
            isLoading && (
              <div className="col-start-1 col-end-12 fade-in">
                <TypingLoader />
              </div>
            )
          }


        </div>
      </div>


      <TextMessageBox
        onSendMessage={ handlePost }
        placeholder='Escribe aquí lo que deseas'
        disableCorrections
      />

    </div>
  );
};
```

- core/use-cases
- create-thread-use-case

```
export const createThreadUseCase = async () => {

  try {

    const resp = await fetch(`${ import.meta.env.VITE_ASSISTANT_API }/create-
thread`,{
      method: 'POST'
    });

    const { id } = await resp.json() as { id: string };

    return id;

  } catch (error) {

    throw new Error('Error creating thread');
  }


}
```

- post-question-.use-case

```
import { QuestionResponse } from '../../../interfaces';


export const postQuestionUseCase = async ( threadId: string, question: string ) =>
{


  try {

    const resp = await fetch(`${ import.meta.env.VITE_ASSISTANT_API }/user-
question`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ threadId, question })
    });


    const replies = await resp.json() as QuestionResponse[];
    console.log(replies);

    return replies;


  } catch (error) {
    console.log(error);
    throw new Error('Error posting question')
  }


};
```

- interfaces/assistant.response

```
// Generated by https://quicktype.io

export interface QuestionResponse {
  role:    string;
  content: string[];
}
```

- presentation/router/router

```
import { Navigate, createBrowserRouter } from 'react-router-dom';
import { OrthographyPage, ProsConsPage, ProsConsStreamPage, TranslatePage,
TextToAudioPage, ImageGenerationPage, AssistantPage, ImageTunningPage,
AudioToTextPage } from '../pages';
```

```
import { DashboardLayout } from '../layouts/DashboardLayout';

export const menuRoutes = [
  {
    to: "/orthography",
    icon: "fa-solid fa-spell-check",
    title: "Ortografía",
    description: "Corregir ortografía",
    component: <OrthographyPage />
  },
  {
    to: "/pros-cons",
    icon: "fa-solid fa-code-compare",
    title: "Pros & Cons",
    description: "Comparar pros y contras",
    component: <ProsConsPage />
  },
  {
    to: "/pros-cons-stream",
    icon: "fa-solid fa-water",
    title: "Como stream",
    description: "Con stream de mensajes",
    component: <ProsConsStreamPage />
  },
  {
    to: "/translate",
    icon: "fa-solid fa-language",
    title: "Traducir",
    description: "Textos a otros idiomas",
    component: <TranslatePage />
  },
  {
    to: "/text-to-audio",
    icon: "fa-solid fa-podcast",
    title: "Texto a audio",
    description: "Convertir texto a audio",
    component: <TextToAudioPage />
  },
  {
    to: "/audio-to-text",
    icon: "fa-solid fa-comment-dots",
    title: "Audio a texto",
    description: "Convertir audio a texto",
    component: <AudioToTextPage />
  },
  {
    to: "/image-generation",
    icon: "fa-solid fa-image",
    title: "Imágenes",
    description: "Generar imágenes",
    component: <ImageGenerationPage />
  },
  {
    to: "/image-tunning",
```

```
    icon: "fa-solid fa-wand-magic",
    title: "Editar imagen",
    description: "Generación continua",
    component: <ImageTunningPage />
  },

  {
    to: "/assistant",
    icon: "fa-solid fa-user",
    title: "Asistente",
    description: "Información del asistente",
    component: <AssistantPage />
  },
];


export const router = createBrowserRouter([
  {
    path: "/",
    element: <DashboardLayout />,
    children: [
      ...menuRoutes.map( route => ({
        path: route.to,
        element: route.component
      })),
      {
        path: '',
        element: <Navigate to={ menuRoutes[0].to } />
      }

    ],
  }
])
```

- chat-template!!!!

```
import { useState } from 'react';
import { GptMessage, MyMessage, TypingLoader, TextMessageBox } from
'../components';

interface Message {
  text: string;
  isGpt: boolean;
}




export const ChatTemplate = () => {

  const [isLoading, setIsLoading] = useState(false);
  const [messages, setMessages] = useState<Message[]>([])
```

```
  const handlePost = async( text: string ) => {

    setIsLoading(true);
    setMessages( (prev) => [...prev, { text: text, isGpt: false }] );

    //TODO: UseCase

    setIsLoading(false);

    // Todo: Añadir el mensaje de isGPT en true


  }



  return (
    <div className="chat-container">
      <div className="chat-messages">
        <div className="grid grid-cols-12 gap-y-2">
          {/* Bienvenida */}
          <GptMessage text="Hola, puedes escribir tu texto en español, y te ayudo
con las correcciones" />

          {
            messages.map( (message, index) => (
              message.isGpt
                ? (
                  <GptMessage key={ index } text="Esto es de OpenAI" />
                )
                : (
                  <MyMessage key={ index } text={ message.text } />
                )

            ))
          }


          {
            isLoading && (
              <div className="col-start-1 col-end-12 fade-in">
                <TypingLoader />
              </div>
            )
          }


        </div>
      </div>


      <TextMessageBox
```

```
          onSendMessage={ handlePost }
          placeholder='Escribe aquí lo que deseas'
          disableCorrections
        />

      </div>
    );
  };
```