



**INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO**

Manual Técnico

Trabajo Terminal TT2020-B002

**Generador de versos musicales en el idioma
inglés por medio de procesamiento de
lenguaje natural y redes neuronales**

Presentan:

Espinosa de los Monteros Lechuga Jaime Daniel
Nava Romo Edgar Adrián
Salgado Gómez Alfredo Emilio

Directores:

Olga Kolesnikova
Ariel López Rojas

Firmas de Directores

Firmado por:

Profesor: Ariel López Rojas

Doctora Olga Kolesnikova

Índice.

Índice	2
1. Presentación	7
2. Resumen	7
3. Introducción	8
4. Objetivo	8
5. Requerimientos mínimos técnicos	9
5.1. Requisitos de hardware de aplicaciones web	9
5.2. Requisitos de red	9
6. Herramientas utilizadas para el desarrollo	10
6.1. Python	10
6.2. HTML	10
6.3. CSS	10
6.4. JavaScript	11
6.5. Flask	11
6.6. Gunicorn	12
6.7. Amazon EC2	12
6.8. Amazon Elastic Beanstalk	12
6.9. Amazon S3	12
7. Diagramas de la aplicación web	13
7.1. Diagrama de casos de uso	13
8. Funcionamiento general del sistema	15
9. Base de datos	16
10. Limpieza de la base de datos	18
11. Desarrollo del modelo	20
12. Creación del modelo	23
12.1. Embedding o incrustación	23
12.2. Bidireccional	23
12.3. Dropout	23
12.4. Densidad	24

12.5. Método de compilación, algoritmo de optimización y métrica de rendimiento	24
12.6. Importación del modelo	25
13. Desarrollo de la aplicación web	26
14. Desarrollo del backend	33
15. Desplegando componentes	39
15.1. Despliegue del backend	39
15.2. Despliegue de la página web	46
16. Configuración de SSL y DNS	51

Índice de figuras.

1.	Diagrama de casos de uso	13
2.	Diagrama general del sistema	15
3.	Diagrama entidad-relación de la base de datos	16
4.	Diagrama entidad-relación de la base de datos procesada	17
5.	Diagrama entidad-relación final	17
6.	Resultados de limpieza en la base de batos	19
7.	Estadísticas de las palabras	21
8.	Tokenizado de las palabras [13]	22
9.	Padding del tokenizado de las palabras	22
10.	Páginas trabajadas	27
11.	Página de bienvenida	28
12.	Formulario	28
13.	Leyenda mostrada	30
14.	Texto generado	30
15.	Archivo de texto descargado	32
16.	Cuadro de status del Backend	36
17.	Cuadro del método Post	36
18.	Código para la creación de una imagen de Docker con las dependencias necesarias para crear un contenedor con la API	39
19.	Archivo YML con instrucciones para AWS Elastic Beanstalk para subir la imagen correspondiente con la API.	40
20.	Tipos de entornos.	40
21.	Crear un entorno de servidor.	41
22.	Elegir tipo de plataforma que utilizará el entorno.	42
23.	Subir el código de la aplicación.	43
24.	Despliegue del entorno.	43
25.	Primera parte de las configuraciones generales de Elastic Beans- talk.	44
26.	Segunda parte de las configuraciones generales de Elastic Beans- talk.	44
27.	Despliegue del entorno.	45
28.	Estatus de la API corriendo con el endpoint configurado.	45
29.	Instalando CLI de Vercel	46
30.	Indicando acciones para el despliegue	46
31.	Detección del tipo de proyecto a desplegar	47
32.	Desplegando el proyecto	47
33.	Desplegando el proyecto	48
34.	Configuración del proyecto desplegado	48

35.	Variable de entorno	49
36.	Aplicación web desplegada	49
37.	Accediendo a la aplicación web desplegada	50
38.	Dirección IP requerida para la configuración del DNS.	51
39.	Nombre requerido para la configuración del DNS.	51
40.	Estado del certificado SSL que se agregará en Cloudflare para que la API pueda funcionar con HTTPS.	52
41.	Configuración final de los registros DNS.	52

Índice de cuadros.

1.	Requisitos hardware	9
2.	Caso de uso 1	14
3.	Caso de uso 2	14
4.	Caso de uso 3	14
5.	Caso de uso 4	14

1. Presentación

El siguiente manual se ha desarrollado con la finalidad de dar a conocer la información necesaria para aquellos que darán soporte a la aplicación web, este les brindara información sobre los requerimientos, el desarrollo de la aplicación web, la generación del modelo, la conexión de la aplicación web con el modelo, las herramientas empleadas y la funcionalidad final.

2. Resumen

El manual detalla aspectos técnicos e informáticos relacionados con el desarrollo de la aplicación web, tiene como finalidad dar a conocer la información necesaria al personal que vaya a administrarlo, modificarlo o para realizar mantenimiento. En este manual se detallan las herramientas que se utilizaron durante el desarrollo.

3. Introducción

Este manual describe los pasos necesarios para que cualquier persona con ciertas bases en sistemas computacionales pueda administrar, editar o configurar la aplicación web, y que cuando lo haga este responda de una manera adecuada.

Se darán a conocer las herramientas que se utilizaron para el desarrollo de la aplicación web, su despliegue, así como se hará apoyo de diagramas e ilustraciones alusivas al funcionamiento del producto. Además se detallarán los requerimientos mínimos de hardware y software para el correcto funcionamiento de la aplicación web.

4. Objetivo

Informar al usuario sobre la estructura y conformación de la aplicación web con el fin de que pueda darle soporte, realizar modificaciones o actualizaciones a la misma, a través de una descripción de los componentes y funcionalidades que lo conforman.

5. Requerimientos mínimos técnicos

5.1. Requisitos de hardware de aplicaciones web

En la tabla siguiente se enumeran los requisitos de hardware mínimos y recomendados para la aplicación web.

Cuadro 1: Requisitos de hardware mínimos y recomendados

Componente	Mínimo	Recomendado
Procesador	Procesador de x86 o x64 bits de doble núcleo de 1,9 gigahercios (GHz) o más con el conjunto de instrucciones SSE2	Procesador de 64 bits de doble núcleo de 3,3 gigahercios (GHz) o más con el conjunto de instrucciones SSE2
Memoria	2 GB de RAM	4 GB de RAM o más
Resolución	Súper VGA con una resolución de 1024 x 768	Súper VGA con una resolución de 1024 x 768

La ejecución de aplicaciones basadas en modelo en un equipo que no cumpla los requisitos recomendados puede producir un rendimiento inadecuado. Además, puede experimentarse rendimiento satisfactorio ejecutando sistemas que usan otra configuración de hardware que los publicados aquí.

Por ejemplo, un sistema con un procesador moderno de cuatro núcleos, velocidad de reloj más baja y más RAM.

5.2. Requisitos de red

Las aplicaciones basadas en modelo están diseñadas para funcionar mejor en redes que tienen los siguientes elementos:

- Ancho de banda superior a 50 KBps (400 kbps)
- Latencia inferior a 150 ms

Tenga en cuenta que estos valores son recomendaciones y no garantizan un rendimiento satisfactorio. Los valores recomendados se basan en los sistemas que usan solicitudes de un formulario con palabras usuales, el resultado podría variar.

6. Herramientas utilizadas para el desarrollo

6.1. Python

Python es un lenguaje de programación orientado a objetos, de alto nivel con semántica dinámica. Sus estructuras de datos integradas de alto nivel, combinadas con el tipado y enlace dinámico, lo hacen muy atractivo para el desarrollo rápido de aplicaciones, así como para su uso en scripts o para conectar componentes ya existentes. La sintaxis simple y fácil de aprender de Python enfatiza la legibilidad y, por lo tanto, reduce el costo de mantenimiento del programa. Python admite módulos y paquetes, lo que fomenta la modularidad del programa y la reutilización del código. [1]

6.2. HTML

Hypertext Markup Language (HTML) es un lenguaje de marcado que define la estructura de una página web y su contenido. HTML consta de una serie de elementos que se utilizan para encerrar o envolver diferentes partes del contenido para que estos se visualicen o actúen de cierta manera. Las etiquetas adjuntas pueden hacer que una palabra o imagen sea un hipervínculo a otro lugar, pueden poner palabras en cursiva, hacer que la fuente sea más grande o pequeña, etc. [2]

HTML5 es la versión más reciente de HTML, la cual integra nuevos elementos, atributos y comportamientos. Permite describir de mejor manera el contenido de la página web, así como mejora su conectividad con el servidor y almacenamiento, posibilita que las páginas web puedan operar sin conexión usando los datos almacenados localmente del lado del cliente, otorga un mejor soporte al contenido multimedia, así como una mejor integración a API's y un mejor diseño usando CSS3. [3]

6.3. CSS

Cascading Style Sheet (CSS) es el lenguaje para describir la presentación de las páginas web, así como hacerlas más atractivas. Permite adaptar la presentación a diferentes tipos de dispositivos. CSS es independiente de HTML y puede ser empleado con cualquier otro lenguaje de marcado basado en XML o SVG. Usando CSS se pueden controlar con precisión cómo se ven los elementos HTML en el navegador, que presentará para las etiquetas de marcado el diseño que cada uno desee. La separación de HTML de CSS facilita el mantenimiento de los sitios, compartir las hojas de estilo entre páginas

y adaptarlas a distintos ámbitos. [4]

Es un lenguaje basado en reglas: cada usuario define las reglas que especifican los grupos de estilos que van a aplicarse a elementos particulares o grupos de elementos de la página web.

Antes de CSS, las etiquetas como fuente, color, estilo de fondo, alineación, borde y tamaño tenían que repetirse en cada elemento de una página web. Ahora con los CSS, podemos definir cómo se van a comportar las etiquetas, al ser guardado en un archivo por separado, esta misma configuración puede usarse en otra página web ahorrando tiempo diseñándola. Además de que CSS provee de mejor y más detallados atributos para cada etiqueta.

6.4. JavaScript

JavaScript es un lenguaje de programación o secuencias de comandos que permite implementar funciones complejas en las páginas web. Estos scripts pueden ser desarrollados en el mismo HTML para que sean ejecutados automáticamente cuando se carga dicha páginas web, estos scripts se proporcionan y ejecutan como texto sin formato. No necesitan una preparación especial ni una compilación para ejecutarse. [5]

JavaScript puede ejecutarse no solo en un navegador, sino también en un servidor, o en cualquier dispositivo que tenga un programa especial llamado JavaScript engine, el cual permite interpretar y ejecutar los scripts.

JavaScript permite crear contenido dinámico dentro de las páginas web, reaccionar ante algunas acciones realizadas por los usuarios como lo son los clics del ratón, el movimiento del puntero o el presionar cierta tecla, permite enviar peticiones al servidor, así como descargar y subir archivos, además es posible obtener y configurar cookies, mostrar mensajes o alertas al usuario.

6.5. Flask

Flask es un mini marco (framework) web, esto es, un módulo de Python el cual permite desarrollar aplicaciones web. No cuenta con un Manejador de Objetos Relacionales u ORM por sus siglas en inglés, pero si cuenta con características como el enrutamiento de URLs y un motor de plantillas. En general es un marco de aplicación web WSGI.

La Web Server Gateway Interface (WSGI) es una especificación que des-

cribe cómo se va a comunicar un servidor web con una aplicación web, y como se pueden llegar a enlazar distintas aplicaciones web para procesar una solicitud o una petición.

6.6. Gunicorn

Gunicorn, también conocido como unicornio verde “Green Unicorn”, es una de las muchas implementaciones de un Web Server Gateway Interface (WSGI) y se usa comúnmente para ejecutar aplicaciones web hechas con Python. Esta implementa la especificación WSGI de frameworks como Django, Flask o Bottle.

6.7. Amazon EC2

Amazon Elastic Compute Cloud (EC2) [6] proporciona una infraestructura de tecnologías de información que se ejecuta en la nube y funciona como un centro de datos que se ejecuta en su propia sede. Es ideal para empresas que necesitan rendimiento, flexibilidad y potencia al mismo tiempo.

Amazon EC2 es un servicio que permite alquilar un servidor o máquina virtual de forma remota para ejecutar aplicaciones.

6.8. Amazon Elastic Beanstalk

Amazon Elastic Beanstalk [7] es un servicio para implementar y escalar servicios y aplicaciones web con diferentes lenguajes de programación, para este proyecto será útil directamente para la administración de la API, entre las ventajas que se tienen en esta herramienta son el control total de los recursos, tiene una administración de escalamiento de acuerdo a las peticiones y recursos necesarios, la infraestructura detecta automáticamente la última versión y la corre por su cuenta y la implementación general es muy sencilla.

6.9. Amazon S3

Amazon Simple Storage Service (S3) [8], como su nombre lo indica, es un servicio web proporcionado por Amazon Web Services (AWS) que proporciona almacenamiento altamente escalable en la nube.

7. Diagramas de la aplicación web

7.1. Diagrama de casos de uso

En el diagrama de casos de uso se detalla el papel que desempeña la aplicación web con el usuario y con el servidor.

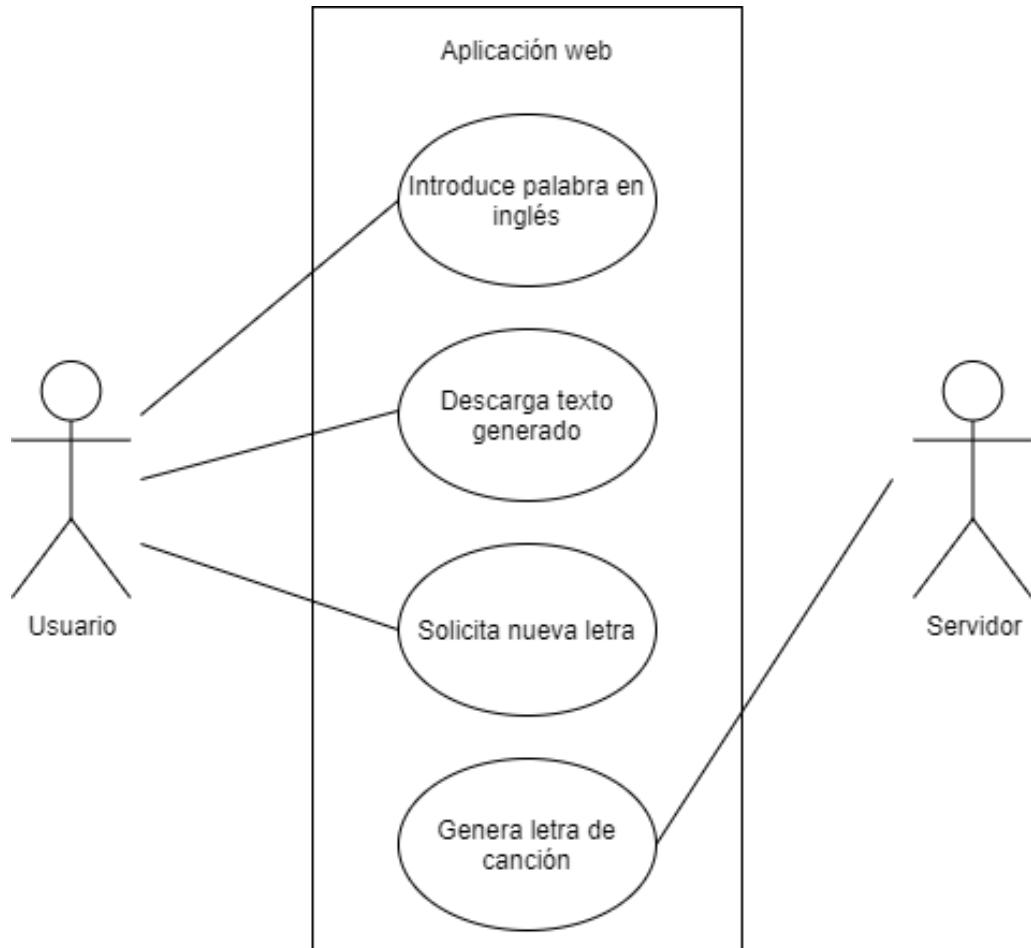


Figura 1: Diagrama de casos de uso

Cuadro 2: Caso de uso 1

Caso de uso	Introduce palabra en inglés
Actores	Usuario
Descripción	El usuario dentro de la aplicación web proporciona una palabra en el idioma inglés para con ella poder generar un texto

Cuadro 3: Caso de uso 2

Caso de uso	Descargar texto generado
Actores	Usuario
Descripción	El usuario dentro de la aplicación web tiene la posibilidad de descargar el texto generado por el modelo

Cuadro 4: Caso de uso 3

Caso de uso	Solicitar nueva letra
Actores	Usuario
Descripción	Al usuario dentro de la aplicación web se le da posibilidad de volver a generar otra letra musical usando los mismos parámetros que proporcionó o utilizando unos nuevos

Cuadro 5: Caso de uso 4

Caso de uso	Genera letra de canción
Actores	Servidor
Descripción	El servidor envía el texto generado por el modelo a la aplicación web, la cual se encarga de mostrarla al usuario final

8. Funcionamiento general del sistema

El siguiente diagrama muestra el funcionamiento general del producto, donde se exemplifica a grandes rasgos la extracción del dataset con el cual, después de limpiarlo se realiza el entrenamiento del modelo, el cual puede generar textos, dichos textos en comunicación con el servidor web se muestran al usuario final.

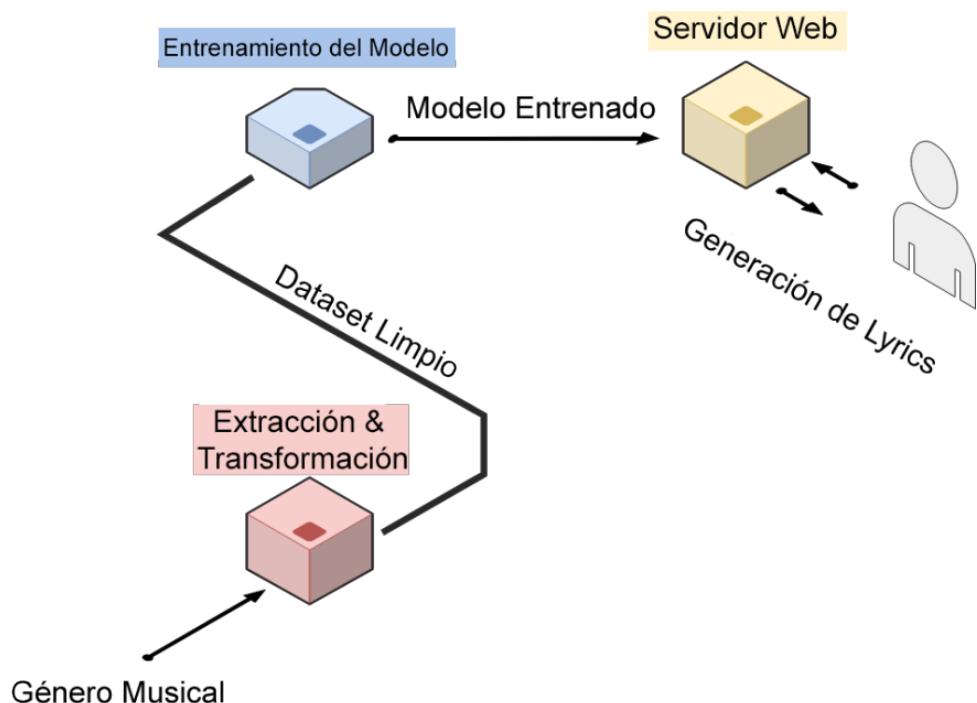


Figura 2: Diagrama general del sistema

9. Base de datos

Lo primero que se realizó fue obtener una base de datos la cual contuviera en ella letras de canciones pertenecientes al género musical con el que íbamos a trabajar, que en este caso fue el género pop, así como que estas se encontraran en el idioma inglés. Para ello buscamos en distintas plataformas, datasets que cumplieran con estos requisitos y que estuvieran disponibles para su uso, es decir, de tipo open source.

En Kaggle [9] se encontró un dataset llamado “Song lyrics for 6 musical genres” [10] el cual contiene todos los datos necesarios para el sistema con un total de 160,790 letras de canciones con las tablas mostradas a continuación:

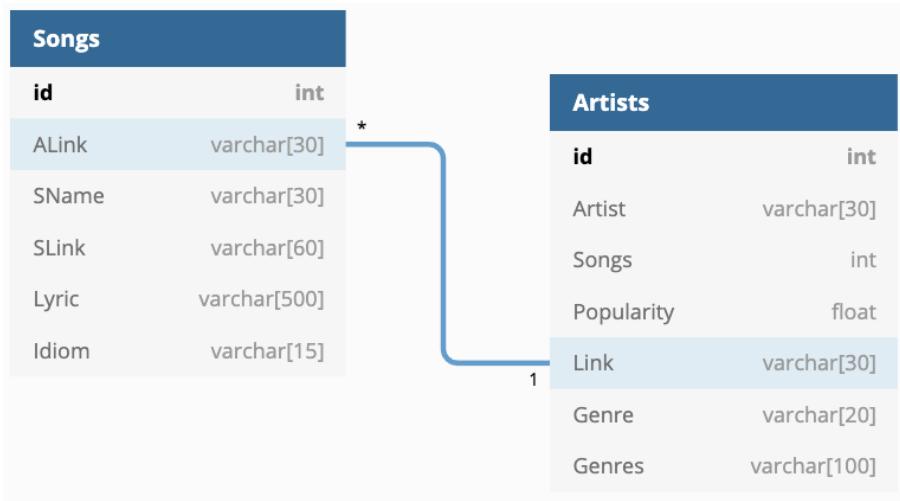


Figura 3: Diagrama entidad-relación de la base de datos

Dicha base de datos se obtuvo originalmente de la página de vagalume.com [11] e incluye 6 tipos de géneros musicales mencionados a continuación.

- Rock
- Pop
- Hip Hop
- Samba
- Sertanejo

■ Funk Carioca

Como se puede observar, se tenía la columna “Genre” la cual fue utilizada para separar el dataset en géneros musicales y “Link” la cual servía como enlace a la segunda tabla. Se fusionaron tablas dando el siguiente resultado:

Songs_with_Artists	
id	int
ALink	varchar[30]
SName	varchar[30]
SLink	varchar[60]
Lyric	varchar[500]
Idiom	varchar[15]
Artist	varchar[30]
Popularity	float
Link	varchar[30]
Genre	varchar[20]
Genres	varchar[100]

Figura 4: Diagrama entidad-relación de la base de datos procesada

Posteriormente es necesario extraer la información precisa que se va utilizar para el entrenamiento del modelo, para ello se utilizaron las columnas de “Lyric” para seleccionar las letras que se utilizaran para entrenar el modelo, “Idiom” para separar las canciones en el idioma inglés y la columna de “Genre” para tamizar las canciones que forman parte del género pop, formando una base de datos simple la cual solo contiene las letras de canciones del género pop en el idioma inglés.

Pop	
id	int
Lyric	varchar[500]

Figura 5: Diagrama entidad-relación final

10. Limpieza de la base de datos

Una vez obtenida la base de datos, es necesario hacer una inspección de esta misma para verificar que los estos datos sean útiles para proceder a generar un modelo, este paso resulta de suma importancia, ya que de esto depende que la precisión de los resultados del modelo.

Para ello se identificarán datos que resulten incompletos, incorrectos, inexactos o no pertinentes para luego sustituirlos, modificarlos o en su caso, eliminar estos datos.

Estas inconsistencias en el conjunto de datos pueden ser causados por errores de entrada del usuario, corrupción de los datos, errores al realizar el scraping, o simplemente cuenta con caracteres que no nos interesa procesar. Una vez realizada esta limpieza, la base de datos será conciliable para usarse en el entrenamiento y así los resultados generados finales tendrán una mejor comprensión.

Para ello se utilizó la librería de “Pandas” así como la librería de expresiones regulares. Para limpiar los datos se siguió el estándar de limpieza de datos: [12]

1. Remover caracteres innecesarios
2. Eliminar Duplicados
3. Evitar errores ortográficos de similitud
4. Convertir los tipos de dato
5. Tratar los valores nulos o faltantes

Con la limpieza del conjunto de datos se tuvo como resultado una base de datos útil solo con los datos necesarios para poder tratarlos con el debido proceso para empezar el entrenamiento del modelo de la red neuronal, este proceso es necesario hacer solo una sola vez para cada género musical ingresado, en caso de que se lleguen a necesitar más datos, será necesario repetir el proceso para poder generar resultados de la manera más óptima posible. A continuación, se muestran las primeras líneas de la base de datos limpias.

	Lyric
0	Hey, slow it down. What do you want from me. W...
1	Died last night in my dreams. Walking the stre...
2	I've paid my dues. Time after time. I've done ...
3	So I got my boots on,. Got the right amount of...
4	I wish that this night would never be over. Th...

Figura 6: Resultados de limpieza en la base de batos

11. Desarrollo del modelo

Para el desarrollo del modelo se utilizó una libreta de Kaggle. Cabe mencionar que el lenguaje de programación utilizado para el desarrollo del modelo fue Python, lo primero que se trabajó en la libreta, fue realizar la importación de librerías que se fueran a necesitar, las más importantes son:

- Pandas: la cual nos permite manipular y analizar la información.
- Wordcloud: es una herramienta que nos ayuda al momento de generar la siguiente palabra de una oración.
- Tensorflow: una herramienta que nos ayuda en el procesamiento del aprendizaje automático.
- Keras: una herramienta que nos ayuda en el procesamiento del aprendizaje profundo.

A continuación, procedemos a importar el conjunto de datos previamente trabajado y limpiado, pero en esta ocasión apoyándonos de la librería de pandas, solo nos vamos a quedar con letra de estas canciones, ya que es la información que nos interesa trabajar y la cual vamos a estar manipulando.

Al usar Kaggle debemos tener en cuenta sus limitantes, en este caso se cuenta solamente con 16Gb de RAM, un disco de 73Gb y una GPU de 13GB para el procesamiento de nuestro conjunto de datos, por eso debemos limitar nuestro modelo, en lugar de utilizar todos los datos (28441), vamos a trabajar solo con 700 letras de canciones por el momento.

Utilizando estas 700 lyrics, se decidió obtener información de ellas, siendo más específicos, estadísticas sobre el número de palabras en cada canción, esto con el fin de determinar la frecuencia de distribución del número de palabras de cada texto y para tener una idea del promedio de palabras, esto con el fin de tenerlo en cuenta al momento de realizar la generación de texto.

count	700.000000
mean	248.065714
std	97.323252
min	31.000000
25%	178.000000
50%	233.000000
75%	309.000000
max	589.000000

Figura 7: Estadísticas de las palabras

Lo que podemos observar en la imagen anterior es:

- Count: el cual es el número de canciones analizadas.
- Mean: el promedio de palabras por canción.
- Std: la desviación estándar de las palabras.
- Min: la menor cantidad de palabras encontradas en una canción.
- Max: la mayor cantidad de palabras encontradas en una canción.

Además, se le realizo una tokenización a las letras de nuestras 700 canciones, esto quiere decir que se separó cada palabra y cada palabra se convirtió en un número. Para este proceso se hizo uso de Keras y la clase Tokenizer(), la cual cuenta con dos métodos importantes:

- `_fit_on_text()`: El cual actualiza el vocabulario interno en función de una lista de textos determinada o, en este caso, la columna "Lyrics", donde cada entrada de la lista será un token.
- `_texts_to_sequences()`: El cual transforma cada texto dentro de la lista de textos proporcionados en una secuencia de números enteros; solo se considerarán las palabras conocidas por el tokenizador.

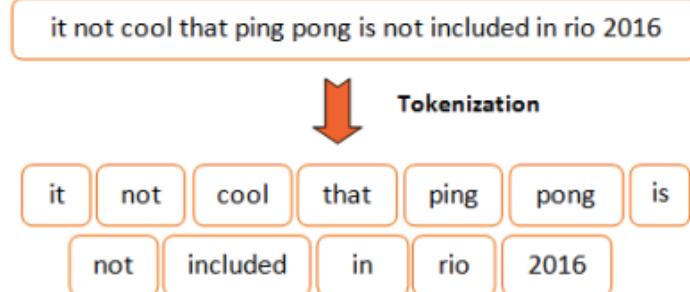


Figura 8: Tokenizado de las palabras [13]

Antes de la generación del modelo, es necesario normalizar todas las oraciones a una misma longitud estándar, para evitar el desbordamiento de la memoria y conseguir que las capas del modelo sean mucho más profundas, este es un proceso simple el cual consta de agregar ceros al comienzo del texto, dando como resultado capas del mismo tamaño.

La posición donde se sumarán los ceros viene determinada por el relleno del argumento, en este caso, se hará al comienzo de la secuencia.

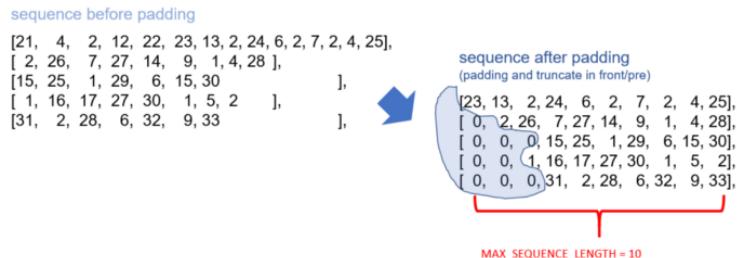


Figura 9: Padding del tokenizado de las palabras

12. Creación del modelo

En este caso se utilizó el modelo LSTM Bidireccional, este tipo de redes neuronales se ejecutan como su nombre lo indica: en dos direcciones. Esto quiere decir que va del pasado al futuro y viceversa, así es como el modelo conserva la información de ambos estados en cualquier momento. Las redes neuronales de LSTM se utilizan principalmente cuando el contexto está involucrado.

Los modelos en Keras se definen como una secuencia de capas, y el modelo secuencial se trata de agregar capas de una en una. Las capas son el componente básico de la red neuronal.

Dentro de estas capas podemos encontrar:

12.1. Embedding o incrustación

Es una capa central, solo se puede usar como la primera capa en un modelo, convierte los números enteros positivos en vectores densos de un tamaño fijo (el primer parámetro es el tamaño del vocabulario, el segundo parámetro es la dimensión de la incrustación densa y el tercer parámetro es sobre la longitud de las secuencias, este se requiere ya que usaremos una capa densa más adelante)

```
1 model.add(Embedding(total_words, 40, input_length=max_sequence_len-1))
```

12.2. Bidireccional

Es una capa recurrente, una envoltura bidireccional para las redes neuronales de tipo RNN's que recibirá una capa como entrada, siendo la capa LSTM la que elegimos, recibirá un entero positivo como entrada que se refiere a la cantidad de nodos de salida que se deben devolver.

```
1 model.add(Bidirectional(LSTM(250)))
```

12.3. Dropout

Es una capa de regularización. Esta capa establece aleatoriamente las unidades de entrada en 0, con una frecuencia del valor que le pasamos, en cada paso durante el tiempo de entrenamiento, lo que ayuda a evitar el sobre ajuste.

```
1 model.add(Dropout(0.1))
```

12.4. Densidad

Es una capa central y una capa de red neuronal densamente conectada. Recibe como primer parámetro un entero positivo que se refiere a la cantidad de nodos de salida que deben devolverse. El segundo parámetro es el llamado activación que define el tipo de predicciones que puede hacer el modelo; para el tipo de problema que estamos considerando, el que se adapta mejor es softmax, que genera un vector de valores (entrada) que puede interpretarse como probabilidades de ser utilizado.

```
1 model.add(Dense(total_words, activation='softmax'))
```

12.5. Método de compilación, algoritmo de optimización y métrica de rendimiento

Perdida: También conocida como función de costos; funciona durante el proceso de optimización y su función es calcular el error del modelo. La entropía cruzada se utiliza para estimar la diferencia entre una distribución de probabilidad estimada y predicha. Se utilizará categorical_crossentropy porque es más adecuado para este tipo de problemas y se usa casi universalmente para entrenar redes neuronales de aprendizaje profundo debido a los resultados que produce. Optimización: Se encarga de reducir las pérdidas y brindar los resultados más precisos posibles. Adam es la opción elegida porque es la mejor opción que ofrece Keras para entrenar la red neuronal en menos tiempo y de manera más eficiente. Earlystop detendrá el entrenamiento si el modelo ha dejado de mejorar, esto se verificará al final de cada epoch. En este caso, la "precisión." "accuracy" se utilizará como métrica de rendimiento. El método fit es el encargado de entrenar el modelo para el número fijo de epochs dados.

```
1 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
2 earlystop = EarlyStopping(monitor='loss', min_delta=0, patience=3, verbose=0, mode='auto')
3 history = model.fit(X, y, epochs=20, verbose=1, callbacks=[earlystop])
```

Quedando como resultado el código del modelo de la siguiente forma:

```

1 model = Sequential()
2 model.add(Embedding(total_words, 40, input_length=max_sequence_len-1))
3 model.add(Bidirectional(LSTM(250)))
4 model.add(Dropout(0.1))
5 model.add(Dense(total_words, activation='softmax'))
6 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
7 earlystop = EarlyStopping(monitor='loss', min_delta=0, patience=3, verbose=0, mode='auto')
8 history = model.fit(X, y, epochs=20, verbose=1, callbacks=[earlystop])

```

12.6. Importación del modelo

Una vez completado el entrenamiento de nuestro modelo, lo que falta es importarlo para probar cómo funciona, en nuestro caso nombramos al modelo “song_lyrics_generator” y se importo de la siguiente forma, llamándola link de nuestra libreta de Kaggle:

```

1 from keras.models import load_model
2 model = load_model('../input/songlyricmodel/song_lyrics_generator.h5')

```

Ya que contemos con el modelo importado, se creó una función que se utilizará para generar la letra de una canción utilizando el modelo previamente entrenado, predecirá las siguientes palabras en base a la palabra(s) de entrada suministradas como ‘seed_text’. Para que esto funcione, se debe aplicar una tokenización al seed_text, luego se aplicará un relleno a las secuencias generadas y se pasará al modelo entrenado para que se pueda predecir la siguiente palabra.

```

1 def complete_this_song(seed_text, next_words):
2     for _ in range(next_words):
3         token_list = tokenizer.texts_to_sequences([seed_text])[0]
4         token_list = pad_sequences([token_list], maxlen=max_sequence_len-1,
5                                     padding='pre')
5         predicted = model.predict_classes(token_list, verbose=0)
6         output_word = ""
7         for word, index in tokenizer.word_index.items():
8             if index == predicted:
9                 output_word = word
10                break
11         seed_text += " " + output_word
12     return seed_text

```

13. Desarrollo de la aplicación web

Para el desarrollo de la aplicación web se decidió realizarla haciendo uso de React o mejor conocida como ReactJS la cual se trata de una biblioteca de Java para crear interfaces interactivas, con la cual se puedes diseñar vistas simples y en las cuales se actualizarán y renderizarán los componentes necesarios cuando haya cambios en los datos.

Se decidió usar este, debido a que nos da la posibilidad de que sea compatible con dispositivos móviles sin la necesidad de volver a escribir código para esta tecnología.

Lo primero que se requirió para poder trabajar con React fue descargar Node.js desde su página (<https://nodejs.org/es/>) la cual nos va a permitir crear aplicaciones web utilizando JavaScript, además de que al instalarlo obtendremos npm el cual posteriormente nos permitirá instalar paquetes con los que añadiremos nuevas funciones a nuestra aplicación y que a su vez son compatibles con React.

Una vez que se tiene instalado Node.js para poder ejecutar una aplicación web, en terminal tendremos que escribir npm start, lo cual ejecutara un servidor de desarrollo de manera local, a la cual se puede acceder a través de la siguiente dirección (localhost:3000) en cualquier navegador.

Como React se basa en crear vistas lo primero que se hizo fue visualizar que es lo que queremos que vea el usuario, por ello se creó una carpeta en la cual se colocaron las páginas con las cuales se iban a trabajar y con las que el usuario va a interactuar. Estas fueron la página de inicio, la de preguntas frecuentes, la de ejemplos de canciones previamente generadas y una acerca de nosotros.

```
✓ components
  > css
  JS AboutLink.js
  JS BasicContainer.js
  JS Button.js
  JS CollapsedListItem.js
  JS Documents.js
  JS Footer.js
  JS Form.js
  JS Header.js
  JS LyricSample.js
  JS MobileMenu.js
  JS Navbar.js
  JS Slider.js
  JS SocialNetworksFooter.js
  JS Swagger.js
  JS TeamContainer.js
✓ pages
  > css
  ✓ js
    JS About.js
    JS Canvas.js
    JS Contact.js
    JS Dev.js
    JS Faq.js
    JS Home.js
    JS Lyricgenerator.js
    JS Samples.js
    JS Team.js
    JS Writing.js
```

Figura 10: Páginas trabajadas

La más importante de estas páginas es la página principal, debido a que es esta la primer a que va a ver el usuario y con la que más va a interactuar, en esta, aparece un mensaje con un botón el cual invita al usuario a generar

su propia letra musical.

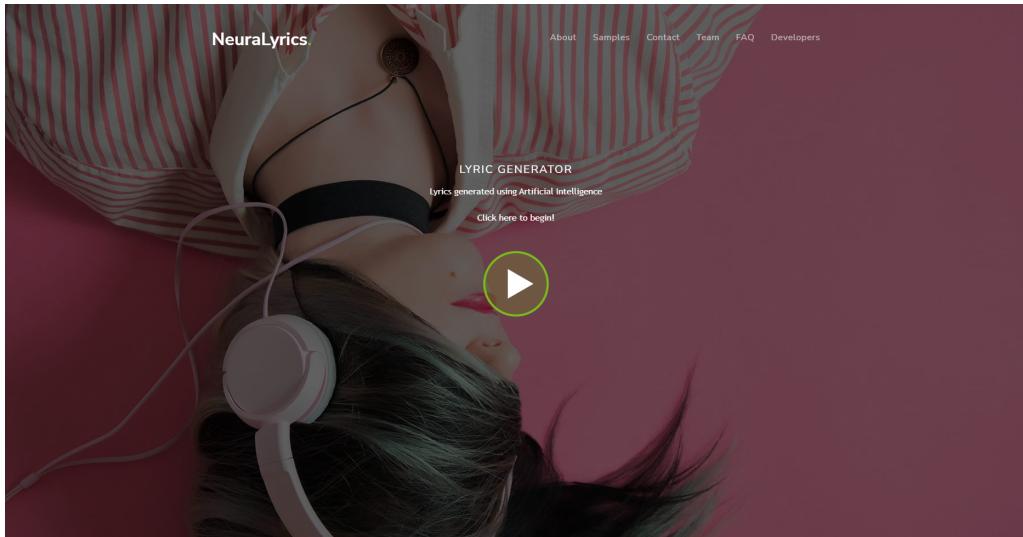


Figura 11: Página de bienvenida

Si el usuario le da clic al botón, como React usa estados para renderizar lo que ve el usuario, se muestra en pantalla el formulario donde se le pide al usuario que introduzca una palabra en el idioma inglés, así como se muestra una barra donde el usuario puede ingresar que tanto quiere que rime la letra de esta canción.

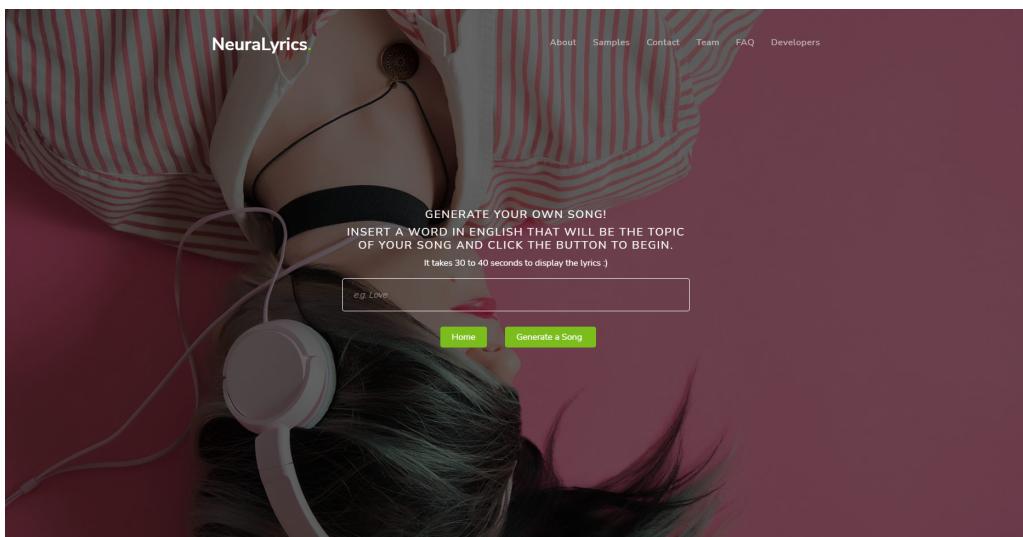


Figura 12: Formulario

En esta parte debemos detenernos para hablar acerca de cómo es que los datos proporcionados por el usuario son leídos y posteriormente enviados a nuestro backend para ser procesados futuramente por el modelo. A continuación, se muestra el código de la función la cual realiza este proceso.

```
1 function sendWord(engword, percentval) {
2
3     localStorage.setItem("EnglishWord-Value", engword);
4     localStorage.setItem("Percentage-Value", percentval);
5
6     fetch('https://api.neuralyrics.com/lyrics', {
7         method: 'POST',
8         headers:{'content-type': 'application/json', 'Access-
Control-Allow-Origin': '*'},
9         body: JSON.stringify({ "lyric_input": engword, "percentage": percentval})
10    ).then(response => {
11        return response.json()
12    }).then(json => {
13
14        var wholeAnswer= JSON.stringify(json);
15        var resp= JSON.parse(wholeAnswer);
16        console.log('Data to work with: ' + resp);
17
18    }).catch(error => {
19        console.log(error);
20    })
21 }
```

En esta función lo primero que se hace es recibir los datos (la palabra y el porcentaje) que introdujo el usuario, esto se hace buscando los identificadores de los elementos de la página y obteniendo sus valores, estos, se enviarán al backend usando la función de fetch, lo que hace dicha función es buscar la url, que en este caso es donde se encuentra alojado nuestro backend y busca el método post de esta, como el método post de nuestro backend recibe datos en formato json, los datos que introdujo el usuario deben ser enviados en este formato.

Una vez que el usuario le dio clic al botón de generar canción se muestra una leyenda la cual le pide al usuario que aguarde unos segundos, que se está trabajando en su letra. Pasados unos segundos después, en pantalla se le mostrará la letra de la canción generada, así como 3 botones, el primero de ellos le permite descargar la letra generada en un archivo de texto, el segundo le permite al usuario generar una nueva canción utilizando los mismos parámetros con los que generó la letra actual y el último lo regresa al estado

anterior para introducir nuevos parámetros y con estos generar una nueva letra.

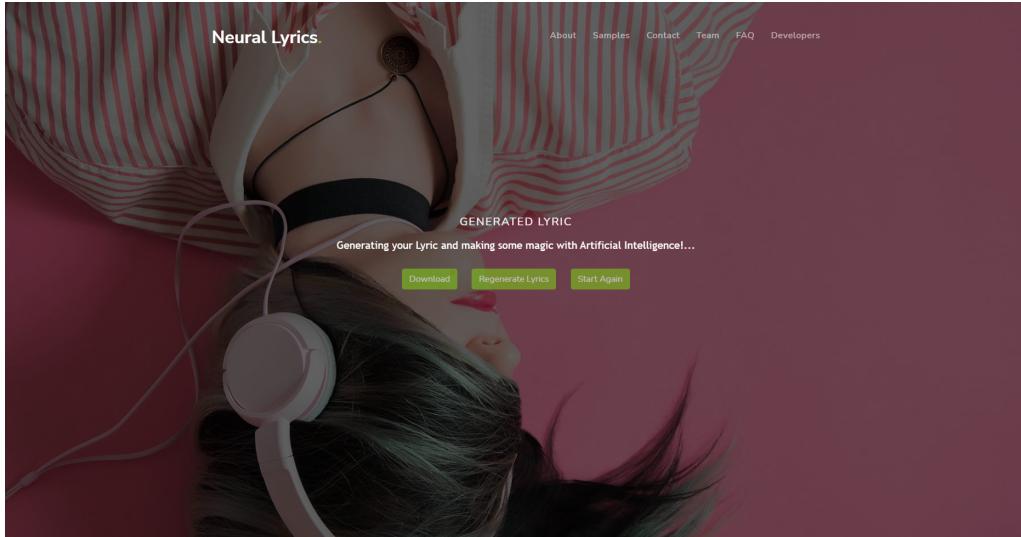


Figura 13: Leyenda mostrada

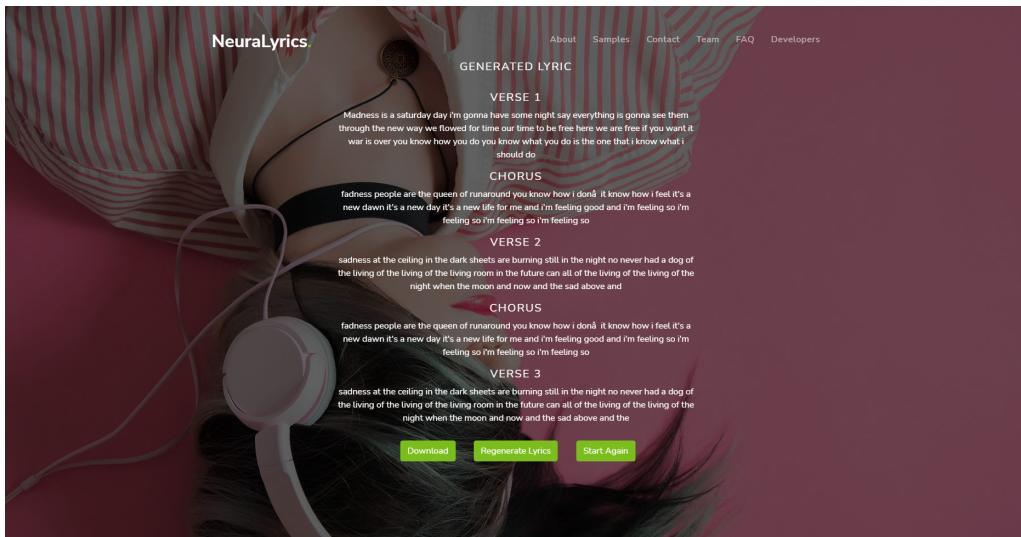


Figura 14: Texto generado

Adentrándonos a esta sección, la primera función que debemos revisar es la que permite hacer la conexión con el backend para que la página pueda recibir el texto generado por el modelo.

```

1 function myFunFactsFunction(){
2     var position= getRandomInt(0, funfacts.length - 1);
3
4     myFunFactsTypewriter('myTextReceived', position, 0);
5 }
6
7 function myFunFactsTypewriter(receivedId, arrayPosition,
8     textPosition){
9     document.getElementById(receivedId).innerHTML = funfacts[
10        arrayPosition].substring(0, textPosition) + '<span>\u25AE
11        </span>';
12
13     if((textPosition++ < funfacts[arrayPosition].length) && !
14        rebooted){
15         setTimeout(myFunFactsTypewriter.bind(null, receivedId
16             , arrayPosition, textPosition), speed);
17     }else{
18         document.getElementById(receivedId).innerHTML =
19         funfacts[arrayPosition].substring(0, textPosition);
20     }
21 }
```

Lo que se hace en el código anterior es realizar una conexión asíncrona con el backend, esto para que mientras no se reciba una respuesta el usuario en pantalla vea una leyenda que diga que se está trabajando en su texto, cuando en la conexión se reciba una respuesta la leyenda que ve el usuario cambiara, mostrando el texto recibido en la función.

```

1 function download(filename, text) {
2     var element = document.createElement('a');
3     element.setAttribute('href', 'data:text/plain;charset
4         =utf-8,' + encodeURIComponent(text));
5     element.setAttribute('download', filename);
6
7     element.style.display = 'none';
8     document.body.appendChild(element);
9
10    element.click();
11
12    document.body.removeChild(element);
13 }
```

Con esta función se permite crear un archivo de texto el cual es posible descargarse en la computadora del usuario y el cual va a contener la letra de la canción que se generó en ese momento.

```
Lyric.txt
VERSE 1
Madness is a saturday day i'm gonna have some night say everything is
gonna see them through the new way we flowed for time our time to be free
here we are free if you want it war is over you know how you do you know

CHORUS

Nattiness people are the queen of runaround you know how i donâ&lt;0x89>it
know how i feel it's a new dawn it's a new day it's a new life for me and
i'm feeling good and i'm feeling so i'm feeling so i'm feeling so i'm
feeling so

VERSE 2

Matrass people are the queen of runaround you know how i donâ&lt;0x89>it
know how i feel it's a new dawn it's a new day it's a new life for me and
i'm feeling good and i'm feeling so i'm feeling so i'm feeling so i'm
feeling so three and i got a little more time

CHORUS

Nattiness people are the queen of runaround you know how i donâ&lt;0x89>it
know how i feel it's a new dawn it's a new day it's a new life for me and
i'm feeling good and i'm feeling so i'm feeling so i'm feeling so i'm
feeling so

VERSE 3

Medas people are the queen of runaround you know how i donâ&lt;0x89>it know
how i feel it's a new dawn it's a new day it's a new life for me and i'm
feeling good and i'm feeling so i'm feeling so i'm feeling so i'm feeling
so three and i got a little more time and
```

Figura 15: Archivo de texto descargado

Para el segundo botón se utiliza la misma función que se ve en la vista previa se reenvía al backend los parámetros utilizados y se renderiza nuevamente la vista con la leyenda de que espere unos segundos. En el caso del tercer botón solo se activa el estado que permite renderizar la parte donde el usuario introduce los datos.

14. Desarrollo del backend

Para el backend se decidió trabajar con una aplicación web desarrollada en Flask, debido a que de esta manera era fácil y rápida de implementar, además que permitía una mejor integración con el modelo y la aplicación web previamente trabajada en React.

Para poder trabajar con una aplicación web en Flask lo primero que se tiene que hacer o que se recomienda hacer es crear un ambiente virtual usando virtualenv esto para separar nuestro Python instalado originalmente en la computadora del que se va a utilizar en la aplicación. Una vez realizado esto, se inició el desarrollo del código.

```
1 # -*- coding: utf-8 -*-
2 import os
3 import pickle
4 import logging
5 from datetime import datetime
6 from random import randint
7
8 # Tensorflow
9 from tensorflow.keras.models import load_model
10
11 # Flask
12 from flask import Flask, jsonify
13 from flask_cors import CORS
14 from flask_restful import Resource, Api
15 from flask_apispec import marshal_with, doc, use_kwargs
16 from flask_apispec.views import MethodResource
17 from flask_apispec.extension import FlaskApiSpec
18 from apispec import APISpec
19 from apispec.ext.marshmallow import MarshmallowPlugin
20 from werkzeug.exceptions import HTTPException
21
22 # Own Libraries
23 from .utils.schemas import HealthSchema, GetLyrics, PostLyrics
24 from .utils.generate_lyrics import GenerateLyric
25 from .utils.constants import FLASK_ENV, VERSION, PROJECT
26
27
28 app = Flask(__name__)
29 api = Api(app) # Flask restful wraps Flask app around it.
30 cors = CORS(app)
31 app.config.update({
32     'APISPEC_SPEC': APISpec(
33         title='Lyric Generator',
34         version='v1',
35         plugins=[MarshmallowPlugin()],
36         openapi_version='2.0',
37     ),
38     'APISPEC_SWAGGER_URL': '/swagger/', # URI to access API Doc JSON
39     'APISPEC_SWAGGER_UI_URL': '/swagger-ui/' # URI to access UI of API Doc
40 })
41 docs = FlaskApiSpec(app)
42 try:
43     tokenizer_variables = {}
```

```

44     tokenizer_variables['model'] = load_model(os.path.abspath('src/
45         song_lyrics_generator.h5'))
46     logging.info("Model loaded")
47     with open(os.path.abspath('src/tokenizer_data.pkl'), 'rb') as f:
48         data = pickle.load(f)
49         tokenizer_variables['tokenizer'] = data['tokenizer']
50         tokenizer_variables['max_len'] = data['max_sequence_len']
51     logging.info("API configuration is ready.")
52 except Exception as e:
53     logging.error(f'Failed to load model:{str(e)}')
54
55 # Endpoints
56 class Lyrics(MethodResource, Resource):
57     """
58     Main API class to call the endpoint to generate Lyrics
59     get:
60         gets constants to check if the api is uploaded correctly
61     post:
62         post the required fields to generate a lyric
63     """
64     @doc(description='Health-check to see the status of the API.', tags=['
65         Lyric Generator Status'])
66     @marshal_with(HealthSchema)
67     def get(self):
68         """
69         Get method to call configurations of the API
70         """
71         logging.info("Getting status of the API.")
72         return {
73             'project': PROJECT,
74             'version': VERSION,
75             'environment': FLASK_ENV,
76             'date': datetime.now(),
77         }
78     @doc(description='Send input to generate Lyric', tags=['Generate Lyric'
79     ])
80     @use_kwargs(GetLyrics, location='json')
81     @marshal_with(PostLyrics)
82     def post(self, **kwargs):
83         """
84         Generate a Lyric with two kwargs readed in json
85         go to schemas.py to see more of this fields.
86         """
87         logging.info(f'Posting: {kwargs}')
88         response = {}
89         lyrics = GenerateLyric(tokenizer_variables, kwargs)
90         response['title'] = lyrics.complete_this_song(randint(2,5),
91         first_verse=True)
92         response['verse_1'] = lyrics.complete_this_song(randint(40,60),
93         first_verse=True)
94         response['chorus'] = lyrics.complete_this_song(randint(30,50))
95         for medium in range(2,4):
96             response['verse_'+str(medium)] = lyrics.complete_this_song(
97             randint(40,60))
98         return jsonify(response)
99
100
101 # Add Endpoints
102 api.add_resource(Lyrics, '/lyrics')

```

```

100 # Add Swagger Documentation
101 docs.register(Lyrics)
102
103
104 # Handling of 404 errors.
105 @app.errorhandler(404)
106 def page_not_found(e):
107     logging.info(f'Resource not found')
108     return jsonify({"error": "resource not found", "code": "404"}), 404
109
110
111 @app.errorhandler(Exception)
112 def handle_error(e):
113     code = 500
114     if isinstance(e, HTTPException):
115         code = e.code
116         logging.warning(f'Posting: {code}')
117
118     return jsonify(error=str(e)), code
119
120
121 # We run the Flask application and, if it is running in a development
122 # environment,
123 # we run the application in debug mode.
124 # to run only ``FLASK_ENV=test FLASK_APP=src.app python -m flask run --host
125 # =0.0.0.0 --port=80``
126 app.run(debug=FLASK_ENV == 'development', host='0.0.0.0')

```

En el código app.py mostrado anteriormente es el código principal de nuestro backend ya que es este el que se despliega en el navegador web, en este lo primero que se visualiza al entrar es información con respecto a esta aplicación como su nombre, versión y los plugins con los que esta cuenta.

Para poder ver un cómo es que realmente funciona el backend se realizó una pequeña interfaz y es posible acceder a ella a través de la siguiente dirección <https://www.neuralyrics.com/developers> en ella podemos encontrar con mayor detalle el status actual de la api, la cual se encuentra ubicada en el espacio nombrado como health.



Figura 16: Cuadro de status del Backend

Un poco más abajo se encuentra el método post de la api, este es el encargado de recibir la información de la aplicación web realizada en React en formato json, para posteriormente enviar esta información al modelo

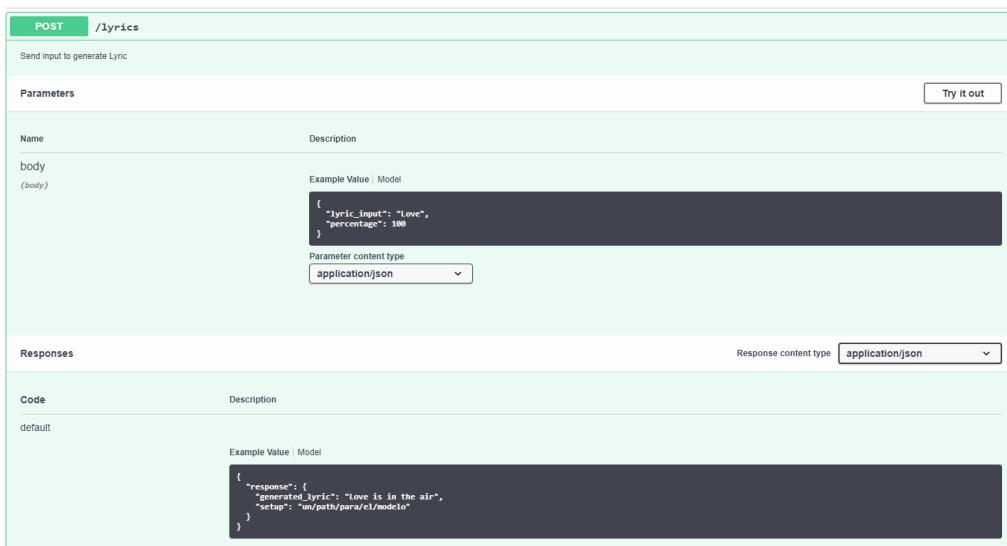


Figura 17: Cuadro del método Post

Para realizar esta tarea se desarrolló el siguiente código

```

1 import requests
2 import random
3 import logging
4 from datetime import datetime
5 from numpy import argmax
6 from tensorflow.keras.preprocessing.sequence import pad_sequences
7

```

```

8
9  class GenerateLyric(object):
10     """
11         Class dedicated to generate new lyrics, only receives
12     """
13     def __init__(self, tokenizer_variables, kwargs):
14         try:
15             self.model = tokenizer_variables['model']
16             self.tokenizer = tokenizer_variables['tokenizer']
17             self.max_len = tokenizer_variables['max_len']
18             self.seed_text, self.percentage = kwargs["lyric_input"], kwargs[
19                 "percentage"]
20             logging.info(f'Receiving {self.seed_text}, {self.percentage}')
21         except Exception as e:
22             logging.error(f'Unexpected error {str(e)}')
23
24     def generate_rhyme(self, first_verse):
25         try:
26             if first_verse is False:
27                 link = f'https://api.datamuse.com/words?sl={self.seed_text}'
28                 rhyme = requests.get(link).json()
29                 if rhyme is not None:
30                     try:
31                         random.seed(str(datetime.now()))
32                         rhyme = random.choice(rhyme)
33                         seed_text = rhyme['word']
34                     except IndexError:
35                         seed_text = self.seed_text
36                 else:
37                     seed_text = self.seed_text
38             return seed_text
39         except Exception as e:
40             logging.error(f'Error processing {self.seed_text}: {str(e)}')
41             seed_text = self.seed_text
42
43     def complete_this_song(self, next_words, first_verse=False):
44         logging.info(f'Generating song.')
45         try:
46             seed_text = self.generate_rhyme(first_verse)
47             for _ in range(next_words):
48                 token_list = self.tokenizer.texts_to_sequences([seed_text])
49             [0]
50                 token_list = pad_sequences([token_list], maxlen=self.max_len
51 - 1, padding='pre')
52                 predicted = argmax(self.model.predict(token_list, verbose=0),
53 , axis=-1)
54                 output_word = ""
55                 for word, index in self.tokenizer.word_index.items():
56                     if index == predicted:
57                         output_word = word
58                         break
59                 seed_text += " " + output_word
60         except AttributeError:
61             seed_text = f'Cannot process {self.seed_text} try with a
62 different word.'
63             logging.error(f'Error processing {self.seed_text}:
64 AttributeError')
65
66         return seed_text, self.percentage

```

En este código utilizando la palabra introducida por el usuario es enviado

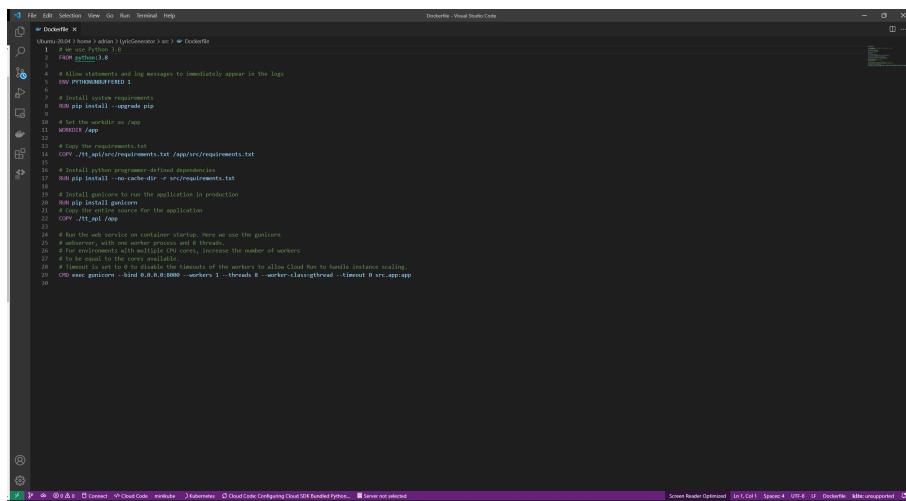
al modelo para ser procesado generando así un texto, este procedimiento se realiza en la función generate_lyric, el texto generado va a ser regresado en forma de json para mandarlo a la aplicación web en React para posteriormente mostrarla en el navegador web del usuario.

15. Desplegando componentes

Una vez lista la comunicación con el modelo, es necesario conectar la función para que pueda recibir llamadas por medio de peticiones web, por lo que se desarrolló una API utilizando Flask para recibir dichas peticiones que en local funcionan llamando al localhost y una vez que se hace la comunicación entre la página y API localmente, se procede a desplegar estos componentes por separado, ya que se manejan diferentes lenguajes de programación y por cuestiones de escalamiento y evitar errores de estructuras monolíticas, se despliega cada componente por separado.

15.1. Despliegue del backend

Para poder desplegar la API, primero se procede a crear una imagen de docker donde se corren todos los comandos y se instalan las dependencias necesarias para poder crear un contenedor con la API y que pueda correr en cualquier sistema operativo compatible con Docker.



```
#!/bin/bash
# Dockerfile
# Dockerfile: A file containing instructions for building a Docker container.
# It typically contains commands to copy files into the container, define environment variables, and specify the entry point for the application.
# This Dockerfile is for a Python application named 'api'.
# It installs Python 3.8 and its dependencies, copies the requirements file, installs them, and runs the application using gunicorn.

# Set the base image to Python 3.8
FROM python:3.8

# Set the environment variable to print logs immediately
ENV PYTHONUNBUFFERED=1

# Install system requirements
RUN pip install --upgrade pip
# Set the working directory to /app
WORKDIR /app
# Copy requirements.txt
COPY ./api/src/requirements.txt ./app/src/requirements.txt
# Install python programmer-defined dependencies
RUN pip install --no-cache-dir -r app/src/requirements.txt
# Install gunicorn to run the application in production
RUN pip install gunicorn
# Copy the application source code
COPY ./api /app
# Run the application
CMD gunicorn app:app
# Run the web service in container scaling. Here we use the gunicorn
# web server provided by our environment to handle multiple worker processes and threads.
# For environments with multiple CPU cores, increase the number of workers
# to match.
# Timeout is set to 60 to disable the timeout of the workers to allow Cloud Run to handle instance scaling.
# CMD exec gunicorn -bind 0.0.0.0:8000 --workers 1 --threads 8 --worker-class gevent --timeout 60 app:app

```

Figura 18: Código para la creación de una imagen de Docker con las dependencias necesarias para crear un contenedor con la API.

Una vez listo el contenedor corriendo localmente, se crea un repositorio con la imagen en un archivo Dockerfile a plataforma <https://hub.docker.com/> para que la herramienta de AWS Elastic Beanstalk descargue la imagen desde ahí con una instrucción de un archivo de Docker Compose como se muestra a continuación.

```

docker-compose.yml ×

Ubuntu-20.04 > home > adrian > LyricGenerator > src > docker-compose.yml > ...
You, a week ago | 1 author (You) | docker-compose.yml (compose-spec.json)
1   version: '3.8'
You, a week ago | 1 author (You)
2   services:
You, a week ago | 1 author (You)
3     neuralyrics:
4       image: adrianromo/neuralyrics:version3
5       ports:
6         - "80:5000"
7

```

Figura 19: Archivo YML con instrucciones para AWS Elastic Beanstalk para subir la imagen correspondiente con la API.

Teniendo estos dos archivos, se procede a crear una cuenta en AWS, se recomienda crear un usuario administrador para que se pueda tener acceso a todas las aplicaciones para configurar las herramientas de cualquier manera, los pasos a seguir para poder crear un entorno en Elastic Beanstalk son los siguientes:

Se selecciona el tipo de entorno que será, en este caso es un entorno web ya que será un API.

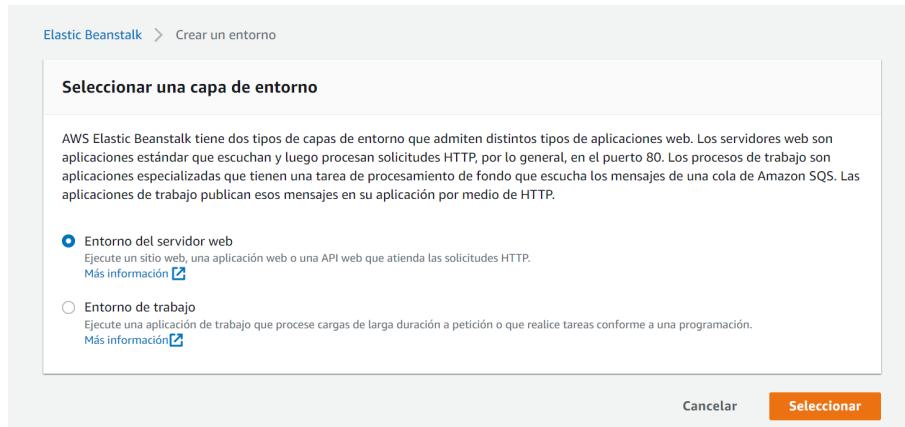


Figura 20: Tipos de entornos.

Se le da un nombre al entorno, se elige un nombre para generar el link a la API y una breve descripción del proyecto.

Crear un entorno de servidor web

Publique un entorno con una aplicación de ejemplo o su propio código. Al crear un entorno, permite que AWS Elastic Beanstalk administre los recursos de AWS y los permisos en su nombre. [Más información](#)

Información de la aplicación

Nombre de la aplicación
Neuralyrics
Hasta 100 caracteres Unicode, sin incluir la barra (/).

► **Etiquetas de la aplicación (opcional)**

Información del entorno

Elija el nombre, el subcampo y la descripción para su entorno. Estas no se pueden cambiar más adelante.

Nombre del entorno
Neuralyrics-env

Campo
Dejar en blanco para generar valor aut. .us-east-1.elasticbeanstalk.
[Comprobar disponibilidad](#)

Descripción
Servicio para generar canciones llamando a un modelo generado.

Figura 21: Crear un entorno de servidor.

Se elige el tipo de plataforma a crear, en este caso se seleccionó como plataforma Docker y se dejaron las configuraciones recomendadas por la misma plataforma.

The screenshot shows the configuration steps for creating a new environment in AWS Elastic Beanstalk. The current step is 'Plataforma' (Platform). The interface includes fields for 'Nombre del entorno' (Environment name), 'Campo' (Namespace), and 'Descripción' (Description). Under the 'Plataforma' section, the 'Plataforma administrada' (Managed platform) option is selected, which is highlighted with a blue border. This option describes platforms published and maintained by Amazon Elastic Beanstalk. There is also a 'Plataforma personalizada' (Custom platform) option, which is described as platforms created by the user. Below these options, there are dropdown menus for 'Plataforma' (Docker), 'Ramificación de la plataforma' (Platform branch) (Docker running on 64bit Amazon Linux 2), and 'Versión de la plataforma' (Platform version) (3.4.9 (Recommended)).

Figura 22: Elegir tipo de plataforma que utilizará el entorno.

Como último paso para crear el entorno, se carga el código en un bucket alojado en S3, este es el archivo con extensión .YML creado previamente para que se pueda crear el entorno a partir de la imagen de docker implementada previamente.

Código de la aplicación

- Aplicación de muestra
Comenzar de inmediato con un código de muestra.
- Versión existente
Versiones de la aplicación que ha cargado para NeuraLyrics.

-- Elija una versión-- ▾

Cargar el código
Cargar un conjunto de fuentes del equipo o copiar uno de Amazon S3.

Etiqueta de versión
Nombre único para esta versión de su código de aplicación.

Origen del código fuente
Tamaño máximo: 512 MB

- Archivo local
- URL de S3 pública

No se ha cargado ningún archivo

► **Etiquetas del código de la aplicación**

Figura 23: Subir el código de la aplicación.

Cuando se le da click al botón de crear entorno, se empezará a cargar la imagen y se debería ver el siguiente estado cuando este listo:



Figura 24: Despliegue del entorno.

En este punto ya se puede ver en el link generado la API, sin embargo hay algunas configuraciones a cambiar que se muestran a continuación del entorno general para que no haya ningún inconveniente al conectar después el Front-end por medio de SSL. En esta primera parte, es importante mencionar que se cambió manualmente el tipo de capacidad de instancia individual a carga balanceada con un mínimo de 1 y máximo de 1, esto no genera ningún cargo extra y nos permitirá después configurar correctamente los certificados para escuchar al puerto 443 de SSL.

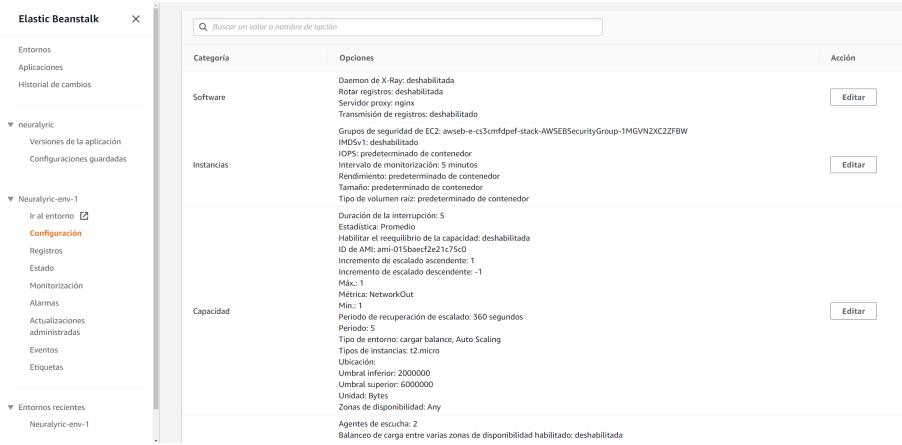


Figura 25: Primera parte de las configuraciones generales de Elastic Beans-talk.

En esta segunda parte, se cambió la configuración del balanceador de carga para que pueda redirigir los puertos de agente de escucha directamente a la API abriendo el puerto de SSL.



Figura 26: Segunda parte de las configuraciones generales de Elastic Beans-talk.

The screenshot shows the AWS Elastic Beanstalk configuration interface for an environment named 'Nealyric-env-1'. The page title is 'Configuración' (Configuration). Under the heading 'Modificar el balanceador de carga clásico' (Modify the classic load balancer), there is a section titled 'Agentes de escucha' (Listeners). A note states: 'Puede especificar agentes de escucha para el balanceador de carga. Cada agente de escucha dirige el tráfico entrante a las instancias en un puerto y con un protocolo especificados. De forma predeterminada, hemos configurado el balanceador de carga con un servidor web estándar en el puerto 80.' (You can specify listeners for the load balancer. Each listener directs incoming traffic to instances on a specific port and protocol. By default, we have configured the load balancer with a standard web server on port 80.)

Puerto	Protocolo	Puerto de instancia	Protocolo de instancia	Certificado SSL	Habilit...
80	HTTP	80	HTTP	--	<input checked="" type="checkbox"/>
443	HTTPS	80	HTTP	*.nealyrics.com - 6b419fdb-3463-409a-b9df-a82059bd9297	<input checked="" type="checkbox"/>

Figura 27: Despliegue del entorno.

Una vez configurado esto, se puede probar que la API este corriendo con los endpoints establecidos en el código de FLASK.

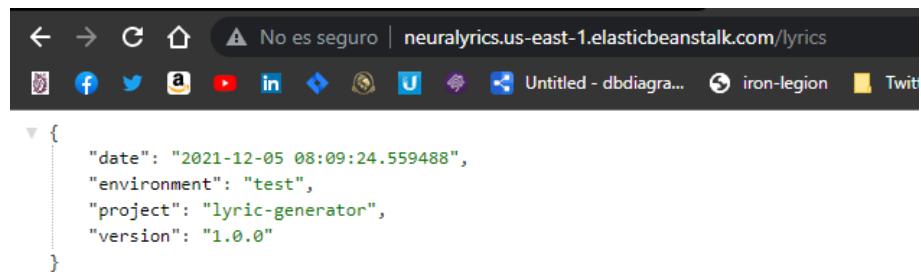


Figura 28: Estatus de la API corriendo con el endpoint configurado.

Una vez lista la API se recomienda hacer pruebas del funcionamiento total de la aplicación generada, y checar los registros que arroja la plataforma de AWS, ya que cualquier error que tenga en este punto la API puede afectar directamente a la página web también, de igual forma se recomienda en este punto crear el certificado que aparece en la parte de Configuración de SSL y DNS para un correcto funcionamiento de la API con HTTPS.

15.2. Despliegue de la página web

Para el despliegue de la pagina web nos apoyamos de la plataforma Vercel, la cual permite que los desarrolladores puedan desplegar sus paginas de manera rápida, así como poder actualizarla y escalarla de manera sencilla, además permite hacer despliegues de proyectos que se encuentran dentro de una cuenta de Github

Lo primero que debemos hacer para poder trabajar con la línea de comandos de Vercel es instalarlo, para ello debemos recurrir a nuestra terminal e instarlo usando el comando npm i -g vercel o yarn global add vercel.

```
PS C:\Users\alfre\Documents\TT2\Vercel\tt_api> npm i -g vercel
> vercel@23.1.2 preinstall C:\Users\alfre\AppData\Roaming\npm\node_modules\vercel
> node ./scripts/preinstall.js

C:\Users\alfre\AppData\Roaming\npm\vc -> C:\Users\alfre\AppData\Roaming\npm\node_modules\vercel\dist\index.js
C:\Users\alfre\AppData\Roaming\npm\vercel -> C:\Users\alfre\AppData\Roaming\npm\node_modules\vercel\dist\index.js
+ vercel@23.1.2
added 96 packages from 110 contributors in 37.665s
PS C:\Users\alfre\Documents\TT2\Vercel\tt_api>
```

Figura 29: Instalando CLI de Vercel

Una vez instalado, debemos abrir la carpeta de nuestro proyecto a desplegar, y en una terminal dentro de esa carpeta solo es necesario escribir vercel para abrir el CLI de Vercel el cual nos preguntará si el proyecto contenido dentro de la carpeta es el que se va a configurar y desplegar, posteriormente nos pide quien lo va a desplegar, para ello usamos una cuenta creada en esta plataforma o vinculamos la de Github para acceder, se nos pregunta el nombre del proyecto y el inicio de los archivos del proyecto a desplegar.

```
PS C:\Users\alfre\Documents\TT2\Vercel2> vercel
Vercel CLI 23.1.2
? Set up and deploy “~\Documents\TT2\Vercel2”? [Y/n] y
? Which scope do you want to deploy to? alfredoesg
? Found project “alfredoesg/vercel2”. Link to it? [Y/n] n
? Link to different existing project? [Y/n] n
? What's your project's name? tt-webp
? In which directory is your code located? ./
```

Figura 30: Indicando acciones para el despliegue

Después de dar las indicaciones anteriores el CLI de vercel detecta automáticamente el tipo de proyecto trabajado, en este caso una aplicación web usando React y por último pregunta si se quiere cambiar la configuración del

trabajo encontrado, en este caso decimos que no, ya que es correcto el tipo de aplicación encontrada con el trabajado.

```
Auto-detected Project Settings (Create React App):
- Build Command: `npm run build` or `react-scripts build`
- Output Directory: build
- Development Command: react-scripts start
? Want to override the settings? [y/N] n
🔗 Linked to alfredoesg/tt-webp (created .vercel)
🔍 Inspect: https://vercel.com/alfredoesg/tt-webp/4hRMKjZD9125a29aCsQXBMc4T7tc [2s]
```

Figura 31: Detección del tipo de proyecto a desplegar

Vercel comienza a desplegar la aplicación web y nos da un enlace en el cual podemos ver como se encuentra el despliegue, si se presentara algún problema, en la terminal nos indicara cual es.

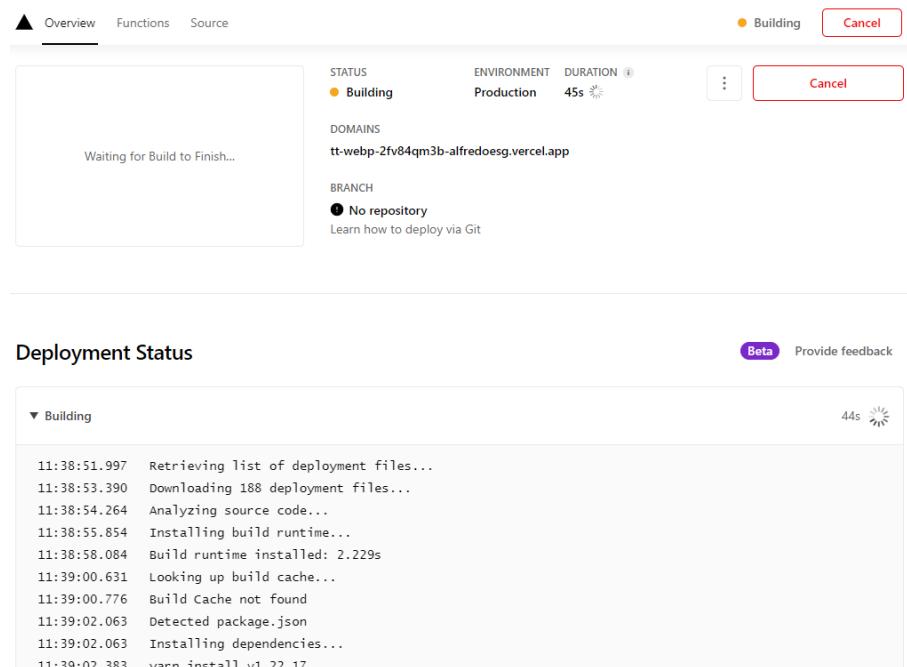


Figura 32: Desplegado el proyecto

Al momento de desplegar la aplicación nos marca un error.

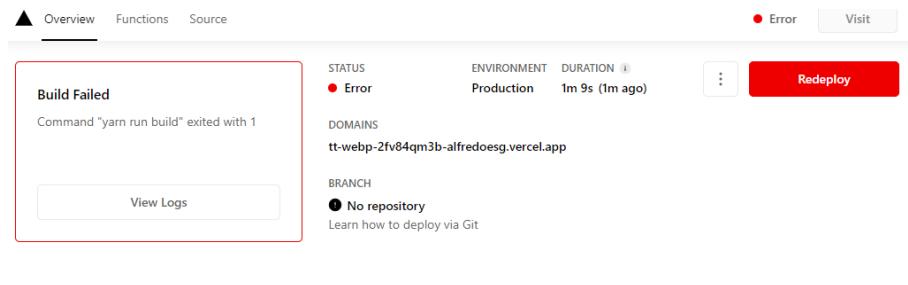


Figura 33: Desplegado el proyecto

Este error está relacionado con unas librerías opcionales localizadas dentro de la carpeta node_modules de nuestro proyecto, vercel no reconoce que estas librerías son opcionales, para poder hacer el despliegue y que no nos marque estas libreríasopcionales como error debemos entrar a la parte de configuraciones de nuestro proyecto.

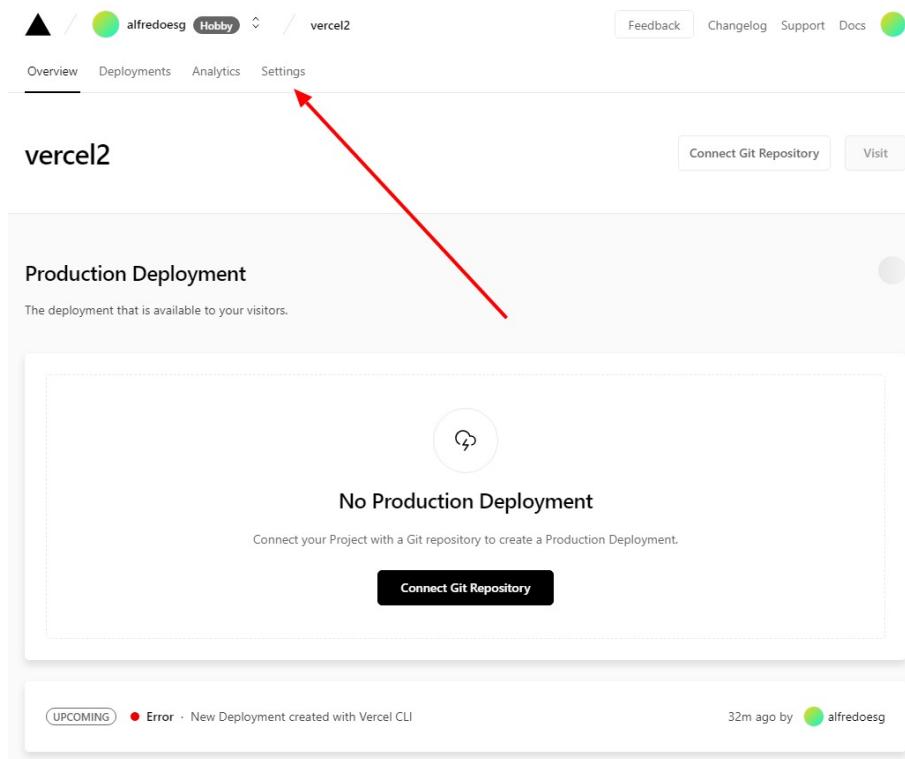


Figura 34: Configuración del proyecto desplegado

Luego en la parte de variables de entorno y en esta sección agregar la variable CI con un valor de false

Project Settings

General **Environment Variables** (highlighted by a red arrow)

Domains

Integrations

Git

Serverless Functions

Environment Variables

Security

Advanced

Add New

NAME	ENVIRONMENT
CI	<input checked="" type="checkbox"/> Production
VALUE	<input checked="" type="checkbox"/> Preview Select Custom Branch
false	<input checked="" type="checkbox"/> Development

It's no longer necessary to create Secrets separately. Just add the value above.

Learn more about [Environment Variables](#)

Add

Figura 35: Variable de entorno

Después de realizar esto volvemos a nuestra terminal, volvemos a escribir vercel y automáticamente vuelve a realizar el despliegue de la aplicación web, de esta misma forma solo escribiendo vercel en la terminal la actualización de la página desplegada ante algún cambio en el código.

Una vez completado el despliegue, se nos genera un enlace con el cual se puede acceder a la aplicación web ya desplegada desde cualquier dispositivo.

Overview **Functions** **Source**

Neural Lyrics

STATUS Ready

ENVIRONMENT Preview

DURATION 1m 15s (1m ago)

DOMAINS tt-webkit-j6b1b7ika-alfredoeg.vercel.app

BRANCH No repository

Learn how to deploy via Git

Visit

Figura 36: Aplicación web desplegada

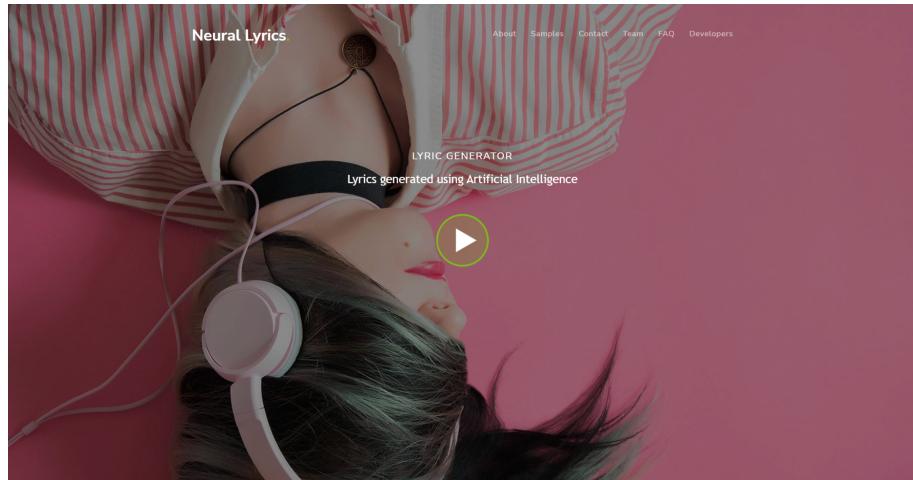


Figura 37: Accediendo a la aplicación web desplegada

En nuestro caso, previamente se había adquirido un dominio, para realizar el cambio del enlace proporcionado por Vercel a el enlace del dominio previamente adquirido, en nuestro proyecto, en la parte de configuración, en la de domino, se agrega el enlace y así es posible acceder a esta pagina web desde ese dominio adquirido.

16. Configuración de SSL y DNS

Para permitir que nuestra pagina web pueda accederse desde otros dispositivos es necesario hacer la configuración correcta de DNS y darle un certificado de seguridad. Debido a que nuestra pagina web fue desplegada en la plataforma de Vercel, para que esta pueda direccionar de manera correcta con nuestro dominio se requiere la configuración de un DNS específico que nos pide Vercel.

You have added the Domain [neuralyrics.com](#) to the Project [tt-webp](#) on [Vercel](#) but it needs to be configured.

To configure the Domain, set the following record on your DNS provider (recommended):

name	type	value
@	A	76.76.21.21

Figura 38: Dirección IP requerida para la configuración del DNS.

You have added the Domain [www.neuralyrics.com](#) to the Project [tt-webp](#) on [Vercel](#) but it needs to be configured.

To configure the Domain, set the following record on your DNS provider (recommended):

name	type	value
www	CNAME	cname.vercel-dns.com .

Figura 39: Nombre requerido para la configuración del DNS.

Por la parte del backend, se utilizó la herramienta gratuita de AWS Certificate Manager, la cuál nos permite agregar un certificado con el puerto número 443 abierto redirigiendo de HTTP a HTTPS, configurándolo para *neuralyrics.com, esto para que se pueda agregar un registro redireccionando a la API y no haya problemas de compatibilidad, una vez que la plataforma haya creado el certificado, se agregan dos registros a Cloudflare:

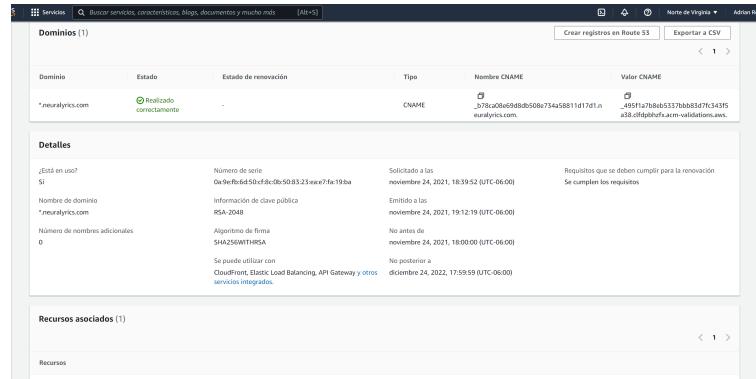


Figura 40: Estado del certificado SSL que se agregará en Cloudflare para que la API pueda funcionar con HTTPS.

Estas reglas de DNS deben agregarse en la plataforma de Cloudflare quedando de la siguiente manera

Tipo	Nombre	Contenido	Estado de proxy	TTL	
A	neuralyrics.com	76.76.21.21	Cloudflare Solo DNS	Automático	Editar
CNAME	api	neuralyrics.us-east-1.elasticbean... información de clave pública	Cloudflare Solo DNS	Automático	Editar
CNAME	_b78ca08e69d8db508e734a...	_495f1a7b8eb5337bb83d7fc34...	Cloudflare Solo DNS	Automático	Editar
CNAME	www	cname.vercel-dns.com	Cloudflare Solo DNS	Automático	Editar

Figura 41: Configuración final de los registros DNS.

1. El primero es el código del certificado creado en AWS de tipo CNAME con el nombre y contenido proporcionado por AWS.
2. El segundo sería otro tipo CNAME con el nombre de 'api' y con contenido a la API directamente.

Una vez Desplegada la parte de la página web y de la API, si se hizo correctamente debería ser posible poder comunicarse por medio de los links generados al final respectivamente, cada parte tendrá su propio versionamiento y como ventaja de esto es que pueden crecer los dos servicios de forma separada sin necesidad de depender forzosamente para que funcione el link una de la otra.

Referencias

- [1] Python (2021), What is Python? Executive Summary, [En línea]. Disponible: <https://www.python.org/doc/essays/blurb/> [Último acceso: 24 de abril del 2021].
- [2] Mozilla.org (2021, febrero 19), HTML basics, [En línea]. Disponible: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics [Último acceso: 15 de mayo del 2021].
- [3] Mozilla.org (2021, mayo 14), HTML5, [En línea]. Disponible: <https://developer.mozilla.org/es/docs/Web/Guide/HTML/HTML5> [Último acceso: 15 de mayo del 2021].
- [4] W3C (2016), HTML & CSS, [En línea]. Disponible: [Último acceso: 15 de mayo del 2021].
- [5] Mozilla.org (2021, abril 27), What is JavaScript?, [En línea]. Disponible: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript [Último acceso: 15 de mayo del 2021].
- [6] Amazon (2021), Amazon EC2 [En línea]. Disponible: <https://aws.amazon.com/es/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc> [Último acceso: 1 Junio 2021]
- [7] Services, A. (2021). Introducción a AWS Elastic Beanstalk. Amazon Web Services, Inc. <https://aws.amazon.com/es/elasticbeanstalk/> [Último acceso: 1 Junio 2021]
- [8] Amazon (2021), Amazon S3 [En línea]. Disponible: <https://aws.amazon.com/es/s3/> [Último acceso: 1 Junio 2021]
- [9] Kaggle (2010), How to Use Kaggle [En línea]. Disponible: <https://www.kaggle.com/docs/datasets> [Último acceso: 25 Mayo 2021]
- [10] Kaggle (2019, Noviembre 17), Song lyrics from 6 musical genres [En línea]. Disponible: <https://www.kaggle.com/neisse/scrapped-lyrics-from-6-genres> [Último acceso: 25 Mayo 2021]
- [11] Vagalume (2002), Vagalume: Music e tudo [En línea]. Disponible: <https://www.vagalume.com.br/> [Último acceso: 25 Mayo 2021]

- [12] Aaron Tay (2021, Abril 23), Data Cleaning Techniques [En línea]. Disponible: <https://www.digitalvidya.com/blog/data-cleaning-techniques/> [Último acceso: 30 Mayo 2021]
- [13] B. Lee (2019, Noviembre 17), How to 10x Response Rates on Surveys [En línea]. Disponible: <https://bryankmlee3.medium.com/conducting-surveys-with-nlp-d38df4c29e39> [Último acceso: 5 Junio 2021]