



**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.

<b>Índice</b>	<b>2</b>
<b>1. Presentación</b>	<b>7</b>
<b>2. Resumen</b>	<b>7</b>
<b>3. Introducción</b>	<b>8</b>
<b>4. Objetivo</b>	<b>8</b>
<b>5. Requerimientos mínimos técnicos</b>	<b>9</b>
5.1. Requisitos de hardware de aplicaciones web . . . . .	9
5.2. Requisitos de red . . . . .	9
<b>6. Herramientas utilizadas para el desarrollo</b>	<b>10</b>
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
<b>7. Diagramas de la aplicación web</b>	<b>13</b>
7.1. Diagrama de casos de uso . . . . .	13
<b>8. Funcionamiento general del sistema</b>	<b>15</b>
<b>9. Base de datos</b>	<b>16</b>
<b>10. Limpieza de la base de datos</b>	<b>19</b>
<b>11. Desarrollo del modelo</b>	<b>21</b>
<b>12. Creación del modelo</b>	<b>24</b>
12.1. Embedding o incrustación . . . . .	24
12.2. Bidireccional . . . . .	24
12.3. Dropout . . . . .	24
12.4. Densidad . . . . .	25

12.5. Método de compilación, algoritmo de optimización y métrica de rendimiento . . . . .	25
12.6. Importación del modelo . . . . .	26
<b>13. Desarrollo de la aplicación web</b>	<b>27</b>
<b>14. Desarrollo del backend</b>	<b>34</b>
<b>15. Desplegando componentes</b>	<b>40</b>
15.1. Despliegue del backend . . . . .	40
15.2. Despliegue de la página web . . . . .	47
<b>16. Configuración de SSL y DNS</b>	<b>52</b>

# 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 . . . . .	18
6.	Resultados de limpieza en la base de batos . . . . .	20
7.	Estadísticas de las palabras . . . . .	22
8.	Tokenizado de las palabras [13] . . . . .	23
9.	Padding del tokenizado de las palabras . . . . .	23
10.	Páginas trabajadas . . . . .	28
11.	Página de bienvenida . . . . .	29
12.	Formulario . . . . .	29
13.	Leyenda mostrada . . . . .	31
14.	Texto generado . . . . .	31
15.	Archivo de texto descargado . . . . .	33
16.	Cuadro de status del Backend . . . . .	37
17.	Cuadro del método Post . . . . .	37
18.	Código para la creación de una imagen de Docker con las dependencias necesarias para crear un contenedor con la API . . . . .	40
19.	Archivo YML con instrucciones para AWS Elastic Beanstalk para subir la imagen correspondiente con la API. . . . .	41
20.	Tipos de entornos. . . . .	41
21.	Crear un entorno de servidor. . . . .	42
22.	Elegir tipo de plataforma que utilizará el entorno. . . . .	43
23.	Subir el código de la aplicación. . . . .	44
24.	Despliegue del entorno. . . . .	44
25.	Primera parte de las configuraciones generales de Elastic Beans- talk. . . . .	45
26.	Segunda parte de las configuraciones generales de Elastic Beans- talk. . . . .	45
27.	Despliegue del entorno. . . . .	46
28.	Estatus de la API corriendo con el endpoint configurado. . . . .	46
29.	Instalando CLI de Vercel . . . . .	47
30.	Indicando acciones para el despliegue . . . . .	47
31.	Detección del tipo de proyecto a desplegar . . . . .	48
32.	Desplegando el proyecto . . . . .	48
33.	Desplegando el proyecto . . . . .	49
34.	Configuración del proyecto desplegado . . . . .	49

35.	Variable de entorno . . . . .	50
36.	Aplicación web desplegada . . . . .	50
37.	Accediendo a la aplicación web desplegada . . . . .	51
38.	Dirección IP requerida para la configuración del DNS. . . . .	52
39.	Nombre requerido para la configuración del DNS. . . . .	52
40.	Estado del certificado SSL que se agregará en Cloudflare para que la API pueda funcionar con HTTPS. . . . .	53
41.	Configuración final de los registros DNS. . . . .	53

## **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 presente manual se ha desarrollado con el objetivo de dar a conocer la información necesaria requerida por quien llegue a dar soporte a la aplicación web. La intención es proporcionar detalles, en un mayor desglose, de los requerimientos seguidos para su elaboración, así como del desarrollo de la aplicación web, la generación del modelo, la conexión entre la aplicación web y el modelo, las herramientas empleadas y la funcionalidad final.

## **2. Resumen**

El manual detalla los aspectos técnicos e informáticos relacionados con el desarrollo de la aplicación web y la finalidad es dar a conocer información necesaria para su manejo u operación al personal que vaya a administrarlo, modificarlo o que tenga como intención el darle mantenimiento, así mismo se exponen en mayor detalle las herramientas utilizadas para su desarrollo.

### **3. Introducción**

El manual describe los pasos requeridos que deben de seguirse para que cualquier operador con nociones básicas de sistemas computacionales pueda administrar, editar o configurar la aplicación web de tal manera que el sistema responda de una manera adecuada y acorde a lo esperado.

Se presentan las herramientas utilizadas para el desarrollo de la aplicación web, así como los pasos que fueron necesarios para la configuración y despliegue de la misma haciendo uso de diagramas e ilustraciones alusivas al funcionamiento como una forma de apoyo visual; se detallan los requerimientos mínimos de hardware y software requeridos para el correcto desempeño de la aplicación web.

### **4. Objetivo**

Dar a conocer al usuario la estructura de la aplicación web y la forma en la cual está conformada con la finalidad de que pueda darle soporte, realizar modificaciones o hacer actualizaciones a la misma.

## 5. Requerimientos mínimos técnicos

### 5.1. Requisitos de hardware de aplicaciones web

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

Cuadro 1: Requisitos mínimos recomendados de hardware

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 la aplicación web en un equipo que no cumpla los requisitos recomendados puede producir un rendimiento inadecuado o no satisfactorio, y de no cumplir con los mínimos puede incluso llevar a un estado de error o de no ejecución.

Haciendo uso del hardware adecuado (superior o igual al recomendado) se podrá percibir un rendimiento satisfactorio y agradable de la aplicación web, esto es, por ejemplo, un sistema con un procesador moderno de cuatro núcleos, velocidad de reloj más baja y más RAM.

### 5.2. Requisitos de red

La aplicación web fue diseñada de tal manera que funciona mejor en redes que cuentan con las siguientes características:

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

Hay que tener en cuenta que estos valores son los recomendados y no garantizan un rendimiento satisfactorio. Los valores sugeridos se tomaron de ejemplos de sistemas que usan solicitudes de un formulario con palabras recurrentes por lo que 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.

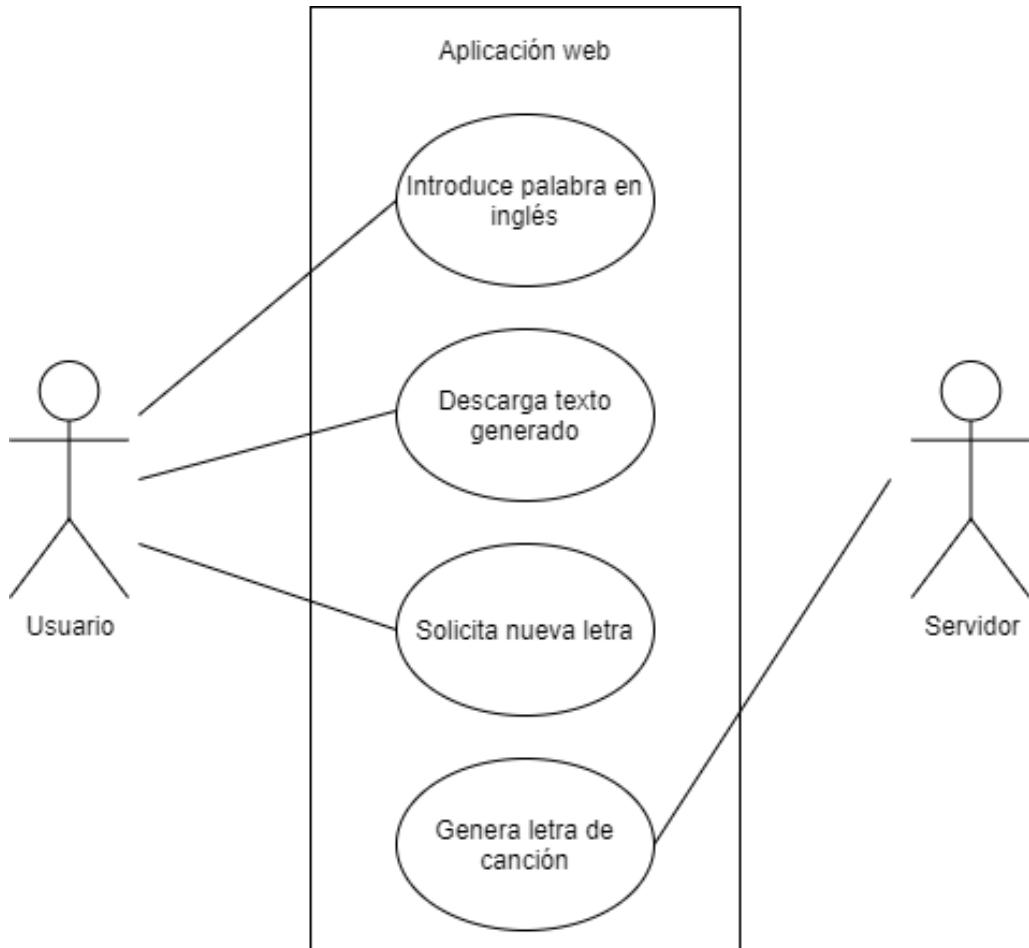


Figura 1: Diagrama de casos de uso

Cuadro 2: Caso de uso 1

Caso de uso	Introducción de palabra en inglés
Actores	Usuario
Descripción	El usuario dentro de la aplicación web proporciona una palabra en el idioma inglés con la finalidad de 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 la posibilidad de volver a generar otra letra musical haciendo uso de la misma palabra que proporcionó o utilizando una nueva.

Cuadro 5: Caso de uso 4

Caso de uso	Generar letra de canción
Actores	Servidor
Descripción	El servidor envía el texto generado por el modelo de vuelta a la aplicación web y ésta se encarga de mostrarla al usuario final.

## 8. Funcionamiento general del sistema

El siguiente diagrama muestra el funcionamiento general del sistema, donde se exponen los pasos seguidos para la realización del mismo, camino que empieza con la extracción del dataset y la limpieza del mismo, el entrenamiento del modelo que con la información obtenida en el paso previo resulta en la generación de textos, los cuales son devueltos por el servidor web y mostrados al usuario final.

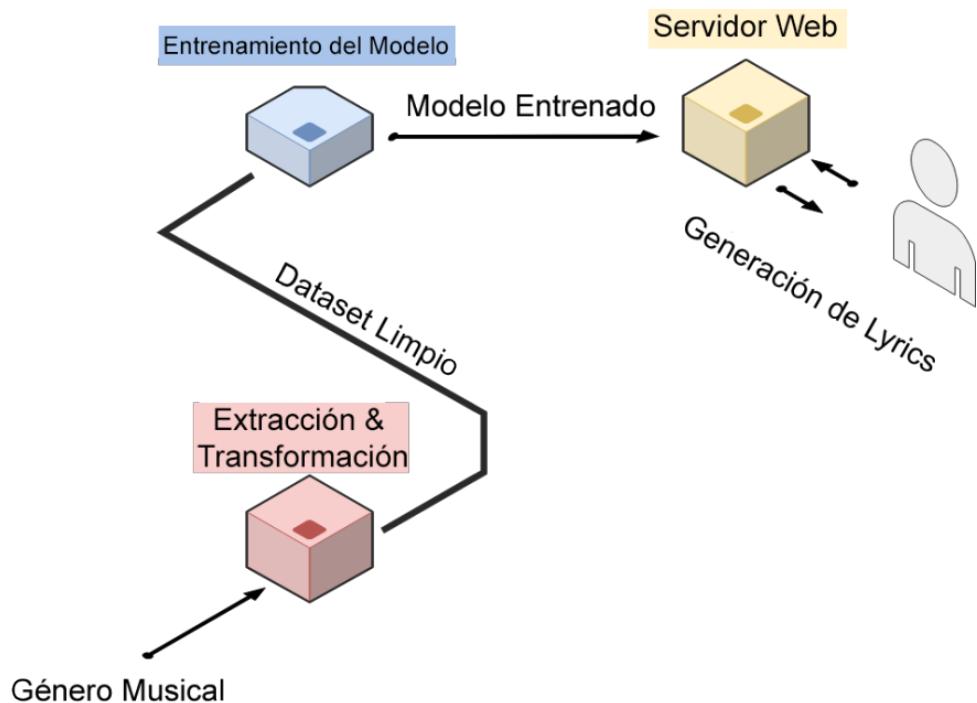


Figura 2: Diagrama general del sistema

## 9. Base de datos

El primer paso realizado fue la obtención de una base de datos que contuviera letras de canciones pertenecientes al género musical con el que íbamos a trabajar, que en este caso fue del género pop, y que se encontraran en el idioma inglés. Se hizo una búsqueda de datasets, en distintas plataformas, que cumplieran con los requisitos ya mencionados y que estuvieran disponibles para su uso libre, es decir, que fueran del tipo open source.

En la plataforma Kaggle [9] se encontró un dataset titulado “Song lyrics for 6 musical genres” [10] el cual contiene todos los datos necesarios y cumplía con los requisitos buscados para el sistema con un total de 160,790 letras de canciones y cuya información se muestra a continuación como se vería en una base de datos, esto es, en tablas:

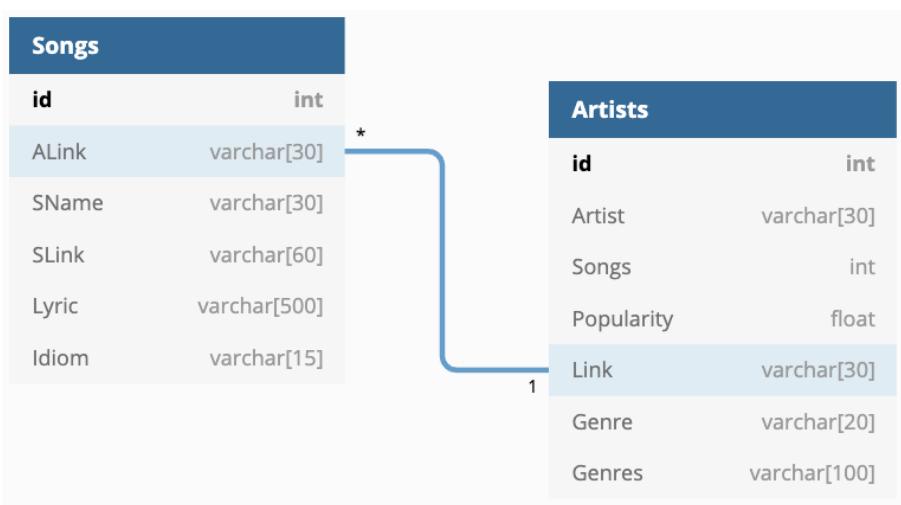


Figura 3: Diagrama entidad-relación de la base de datos

La base de datos de Kaggle 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 ver, se contaba con la columna “Genre” la cual fue utilizada para separar el dataset en géneros musicales y “Link” la cual se usó como enlace a la segunda tabla. Las tablas fueron fusionadas dando como resultado la siguiente:

Songs_with_Artists	
<b>id</b>	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

En este punto se requiere extraer la información precisa que se va utilizar para el entrenamiento del modelo, para ello se hace uso de la columna “Lyric” y así seleccionar las letras que se usarán 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 así una base de datos simple que contiene únicamente las letras de canciones del género pop en el idioma inglés.

Pop	
<b>id</b>	<b>int</b>
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.

	<b>Lyric</b>
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. El lenguaje de programación utilizado para el desarrollo del modelo fue Python, y lo primero que se hizo en la libreta de Kaggle fue la importación de las librerías que se fueran a necesitar, siendo las más importantes:

- Pandas: permite manipular y analizar la información.
- Tensorflow: herramienta que ayuda en el procesamiento del aprendizaje automático.
- Keras: herramienta que ayuda en el procesamiento del aprendizaje profundo.

Llegados a este punto se procede a importar el conjunto de datos previamente limpiado, apoyándonos ahora de la librería de Pandas y conservando únicamente la letra de las canciones, ya que es la información con la que nos interesa trabajar y la cual vamos a estar manipulando.

Al usar Kaggle debemos tomar en consideración sus limitantes, siendo algunas de ellas el solamente contar con 16Gb de RAM, un disco de 73Gb y una GPU de 13GB para el procesamiento de nuestro conjunto de datos, razón por la cual debemos limitar nuestro modelo y así en lugar de utilizar todos los datos (28,441), por el momento sólo podemos trabajar de manera estable con 700 letras de canciones.

Haciendo uso de esas 700 letras se decidió obtener información específica, como el número de palabras en cada canción con el fin de determinar la frecuencia de distribución del número de palabras de cada texto, y así tener una idea del promedio de palabras al momento de realizar la generación de texto.

<b>count</b>	<b>700.00000</b>
<b>mean</b>	<b>248.065714</b>
<b>std</b>	<b>97.323252</b>
<b>min</b>	<b>31.000000</b>
<b>25%</b>	<b>178.000000</b>
<b>50%</b>	<b>233.000000</b>
<b>75%</b>	<b>309.000000</b>
<b>max</b>	<b>589.000000</b>

Figura 7: Estadísticas de las palabras

Lo que podemos observar en la imagen anterior es:

- Count: 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.

Aunado a esto, se le realizó una tokenización a las letras de nuestras 700 canciones, esto significa que se separó cada palabra, y cada una se convirtió en un número. Este proceso se llevó a cabo con el uso de Keras y la clase Tokenizer(), la cual cuenta con dos métodos importantes:

- `_fit_on_text()`: 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()`: 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.

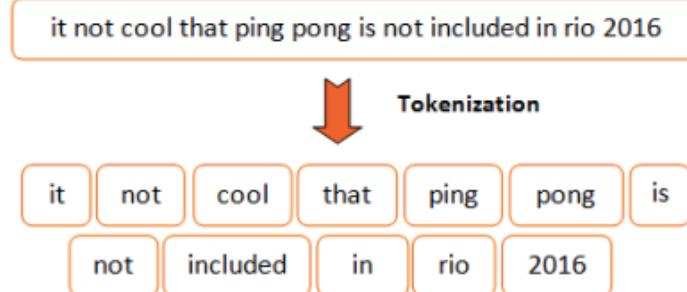


Figura 8: Tokenizado de las palabras [13]

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 así que las capas del modelo sean mucho más profundas, este es un proceso simple conocido como “Padding”, el cual agrega ceros al comienzo del texto dando como resultado capas del mismo tamaño.

La posición donde se agregarán los ceros viene determinada por el “Padding”, en este caso, se hará al comienzo de la secuencia.

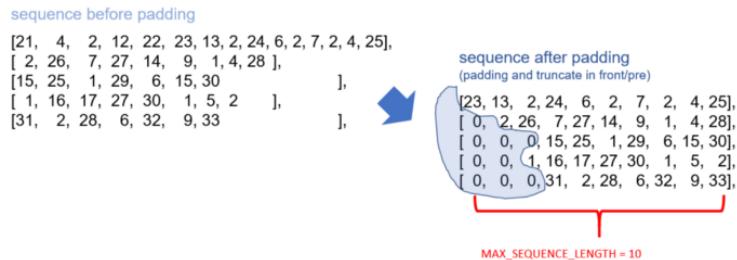


Figura 9: Padding del tokenizado de las palabras

## 12. Creación del modelo

Se utilizó un modelo del tipo 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 del tipo LSTM se utilizan principalmente cuando el contexto importa.

Los modelos en Keras se definen como una secuencia de capas, y las capas son el componente básico de la red neuronal. Algunos ejemplos de tales capas son:

### 12.1. Embedding o incrustación

Es una capa central, solo es posible usarla 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 trata la longitud de las secuencias, este se requiere ya que se hará uso de una capa del tipo Denso 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

“Loss”: También conocida como función de costos; funciona durante el proceso de optimización y su trabajo 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.

“Optimizer”: Su tarea es reducir las pérdidas y brindar los resultados más precisos posibles. Adam es la opción elegida debido a que es la mejor opción que ofrece Keras para entrenar la red neuronal en el menor tiempo y de la forma más eficiente.

“Earlystop”: Detendrá el entrenamiento si el modelo ha dejado de mejorar, y esto es comprobado al final de cada “epoch” o iteración.

En este caso, la “precisión” o “accuracy” se utilizará como métrica de rendimiento. “fit”: Encargado de entrenar el modelo por un 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])
```

El código del modelo queda entonces 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, sólo queda importarlo y así probar su funcionamiento, en nuestro caso nombramos al modelo “song\_lyrics\_generator” y se importó de la siguiente manera, llamando al enlace de nuestra libreta de Kaggle:

```

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

```

Una vez que contamos con el modelo importado se crea una función que se usará para generar la letra de una canción utilizando el modelo previamente entrenado, y éste predecirá las siguientes palabras tomando como contexto a la(s) palabra(s) de entrada suministradas como “seed\_text”. Para que esto funcione, se debe aplicar una tokenización al seed\_text, posterior a ello se aplicará un padding a las secuencias generadas y se pasarán al modelo entrenado, obteniendo de esta manera 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')
6         predicted = model.predict_classes(token_list, verbose=0)
7         output_word = ""
8         for word, index in tokenizer.word_index.items():
9             if index == predicted:
10                 output_word = word
11                 break
12         seed_text += " " + output_word
13     return seed_text

```

## **13. Desarrollo de la aplicación web**

El desarrollo de la aplicación web fue llevado a cabo haciendo uso de React, mejor conocida como ReactJS, la cual es una biblioteca de Javascript usada en mayor parte para la creación de interfaces interactivas diseñadas sobre componentes, los cuales una vez renderizados son mostrados al usuario final.

La biblioteca fue escogida debido a que hace posible la compatibilidad con dispositivos móviles sin la necesidad de escribir código específico para esta tecnología, esto es, nos ofrece un estado responsivo que cambie en función del dispositivo donde se muestre la aplicación web.

Previo a empezar el trabajo con React fue necesario descargar “Node.js” desde su página (<https://nodejs.org/es/>) el cual permite crear aplicaciones web utilizando Javascript, y al haberlo instalado se obtiene el comando “npm” usado en la terminal, el cual permite instalar paquetes al proyecto sobre el que se trabaje posibilitando la adición de nuevas funciones a nuestra aplicación web.

Teniendo instalado Node.js y haciendo uso de una terminal es necesario escribir el comando “npm start”, el cual ejecuta un servidor de desarrollo de forma local al que se puede acceder colocando la dirección “localhost:3000” en cualquier navegador, donde se mostrará cómo va quedando el proyecto sobre el que se esté trabajando.

Para el diseño de la aplicación web nos enfocamos en que su estructura se trabajara bajo el entendido de que las pestañas que se fueran a mostrar sean aquellas que consideramos como las más relevantes, siendo estas: Página de Inicio, Preguntas Frecuentes, Ejemplos de letras de canciones previamente generadas y una acerca del equipo.

```
✓ 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

Entre todo el diseño, la página principal es la más importante debido a que esta es la primera que ve el usuario al acceder a la aplicación web y con la que más va a interactuar durante su estadía; lo primero que se muestra es

un mensaje con un botón verde, colocado con clara intención de invitar al usuario a presionarlo.

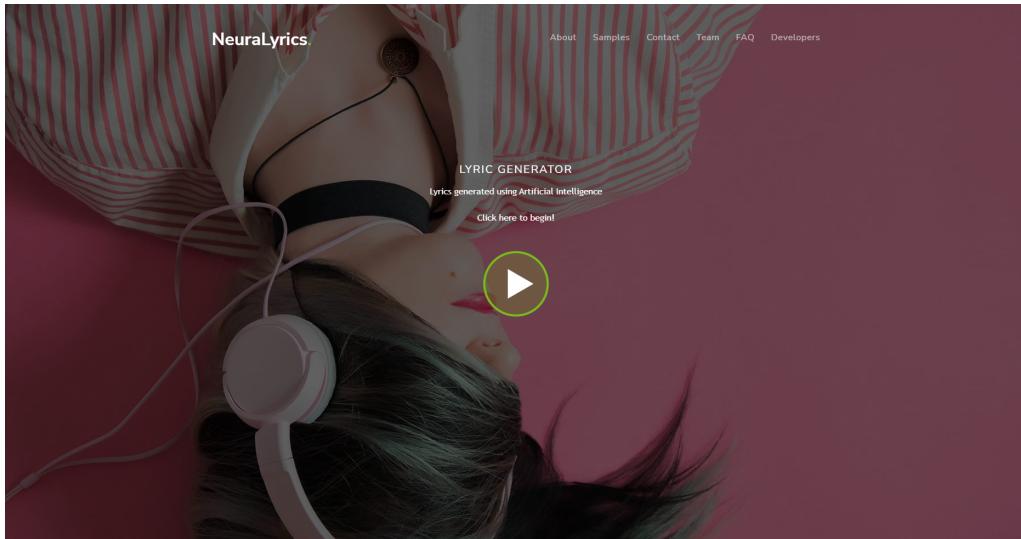


Figura 11: Página de bienvenida

Si el usuario decide presionar el botón será llevado a un formulario sin necesidad de pasar a otra página, maximizando así la sensación de no demora, donde inmediatamente se le es requerido que introduzca una palabra en el idioma inglés en el único campo de entrada de texto, teniendo así la posibilidad de regresar al inicio o de generar la letra con la palabra proporcionada.

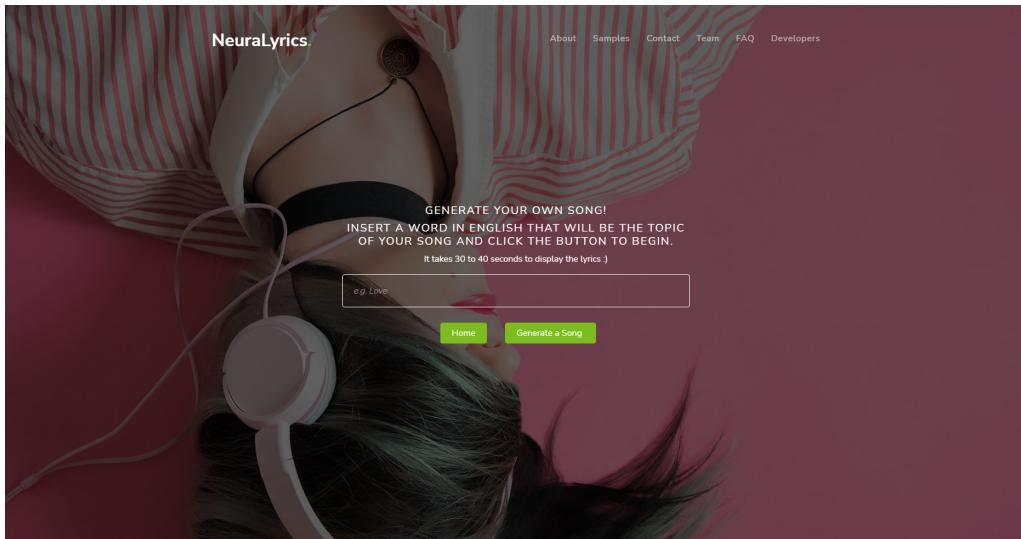


Figura 12: Formulario

Se hace una pausa para mostrar la manera en que el dato proporcionado por el usuario es leído y enviado al servidor para ser procesado por el modelo, y así un resultado más personalizado pueda ser devuelto. A continuación, se presenta el código utilizado para la realización de 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 }
```

Lo que recibe la función es la palabra que introdujo el usuario, esto es hecho ubicando el identificador del campo de texto y obteniendo su valor, el cual es enviado al servidor mediante el uso de la función “fetch”, lo que hace dicha función es buscar la url que se le proporciona (donde se encuentra alojado el servidor) y busca el método post el cual espera recibir los datos en formato Json; los datos proporcionados por el usuario deben ser enviados al servidor en formato json.

Si el usuario decide presionar el botón de Generar Canción será desplegada una leyenda que pide aguardar unos segundos en lo que queda generada la letra. Segundos después, la letra de la canción generada irá siendo actualizada y mostrada al usuario dando un efecto de ser escrita al momento; aparecerán al final de la letra 3 botones, el primero permite descargar la letra generada en un archivo de texto, el segundo posibilita al usuario el generar una nueva canción utilizando la misma palabra con la que generó la letra actual pero esperando un resultado distinto, y el último botón regresa a la

interfaz previa si lo que se busca es generar una nueva letra con una palabra distinta a la ya usada.

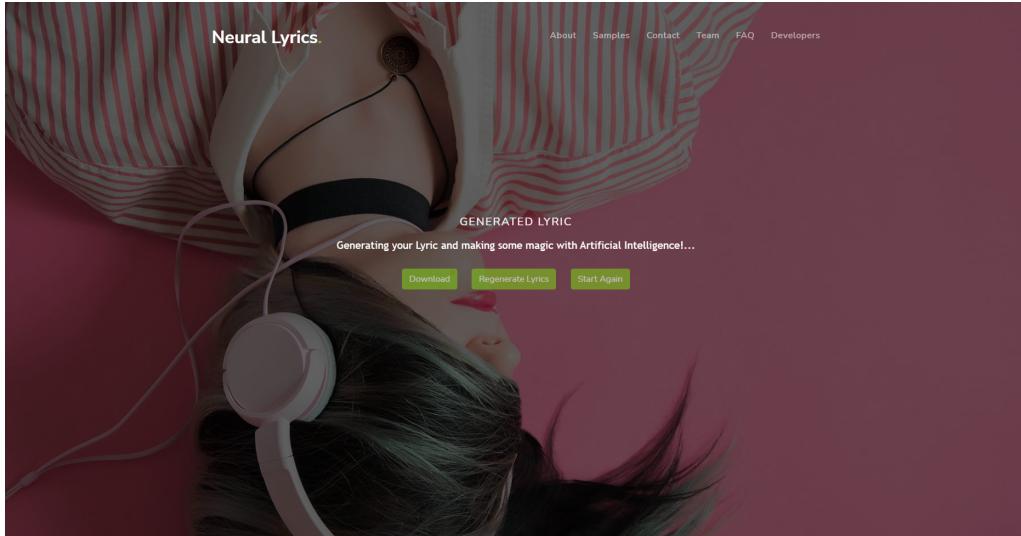


Figura 13: Leyenda mostrada

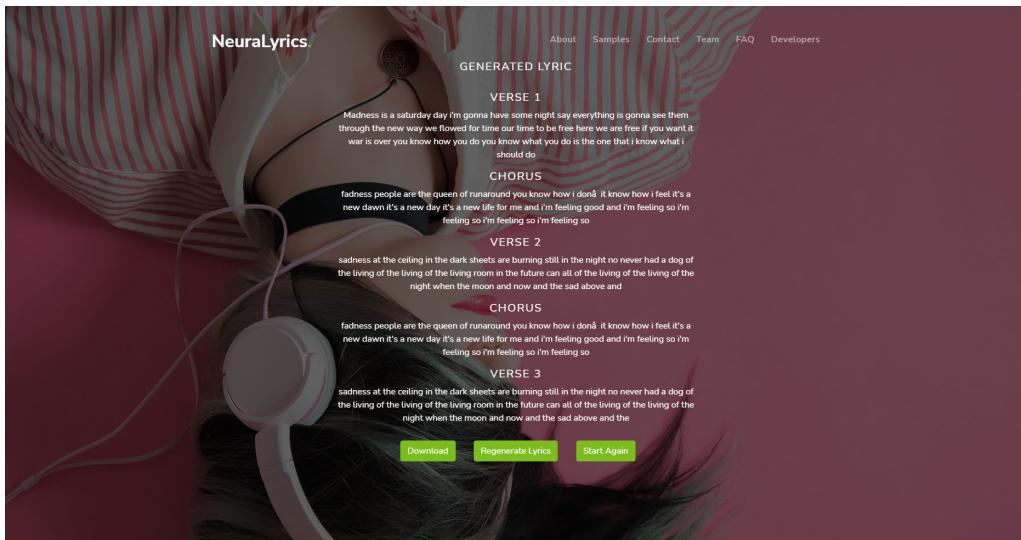


Figura 14: Texto generado

```

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 }
```

El código anterior tiene como objetivo el actualizar la pantalla con una serie de textos pensados para la espera, de manera tal que el usuario vea que se sigue trabajando y no piense que el sitio se congeló, una vez que la letra es recibida se detendrá la generación de datos curiosos y se desplegarán los versos y coros.

```

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 }
```

La función “download” permite crear un archivo de texto contenido la letra de la canción que se generó en ese momento y descargarla al dispositivo que se esté usando.

```
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<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<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<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<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

El segundo botón hace uso de la misma función y el mismo parámetro con los que se genera la letra, esto es, se reenvía al backend los parámetros utilizados y los elementos dentro de la ventana son renderizados nuevamente mostrando una vez la serie de datos curiosos en lo que se recibe respuesta del servidor.

El tercer botón tiene como función regresar al usuario al campo de texto donde puede ingresar la palabra con la que quiere que se genere la letra de la canción.

## 14. Desarrollo del backend

Se tomó la decisión de trabajar con una aplicación web desarrollada en Flask, esto debido a que la implementación es relativamente fácil y rápida de lograr aunado a que permite una mejor integración con el modelo y la aplicación web trabajada en React.

Para poder trabajar con una aplicación web en Flask lo primero que se requiere hacer, o que se recomienda, es crear un ambiente virtual usando “virtualenv” con la finalidad de separar las librerías Python instaladas originalmente en la computadora de las que se requieren por el proyecto con el que se va a trabajar. Una vez se tiene esto preparado, se procede al 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
79         @doc(description='Send input to generate Lyric', tags=[Generate Lyric
80             ])
81         @use_kwargs(GetLyrics, location='json')
82         @marshal_with(PostLyrics)
83         def post(self,**kwargs):
84             """
85                 Generate a Lyric with two kwargs readed in json
86                 go to schemas.py to see more of this fields.
87             """
88             logging.info(f'Posting: {kwargs}')
89             response = {}
90             lyrics = GenerateLyric(tokenizer_variables, kwargs)
91             response['title'] = lyrics.complete_this_song(randint(2,5),
92                 first_verse=True)
93             response['verse_1'] = lyrics.complete_this_song(randint(40,60),
94                 first_verse=True)
95             response['chorus'] = lyrics.complete_this_song(randint(30,50))
96             for medium in range(2,4):
97                 response['verse_'+str(medium)] = lyrics.complete_this_song(
98                     randint(40,60))
99             return jsonify(response)
100
101     # Add Endpoints

```

```

98     api.add_resource(Lyrics, '/lyrics')
99
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')

```

El código anterior se puede encontrar en el archivo “app.py”, es el código principal de nuestro backend y se despliega en el navegador web; muestra información relacionada a la aplicación que se está trabajando como su nombre, versión y los plugins con los que esta cuenta.

Con el objetivo de dar mayor claridad a la manera en que funciona el backend, se realizó una pequeña interfaz a la que es posible acceder desde la siguiente dirección “<https://www.neuralyrics.com/developers>” y en ella podemos encontrar desglosado en mayor detalle el status actual de la API, la cual se encuentra ubicada en el espacio que lleva por nombre “Health”.

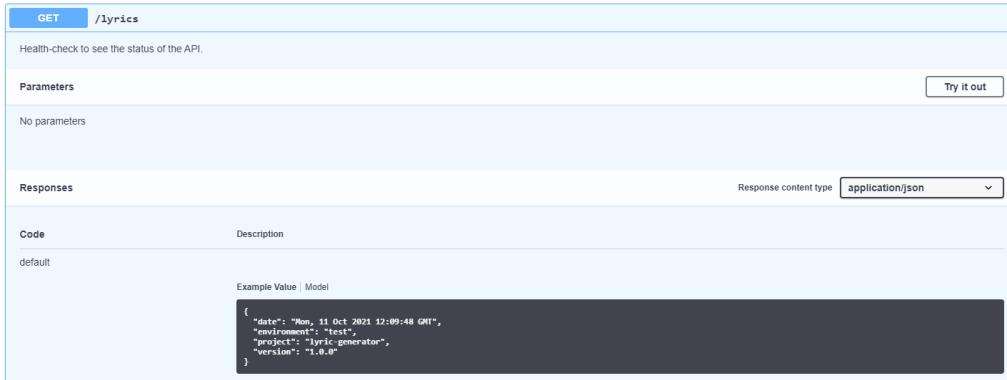


Figura 16: Cuadro de status del Backend

Yéndonos más abajo dentro de la misma dirección se puede encontrar el método post de la API, el cuál tiene como tarea el recibir la información de la aplicación web realizada en React en formato json, y así enviar esta información al modelo.

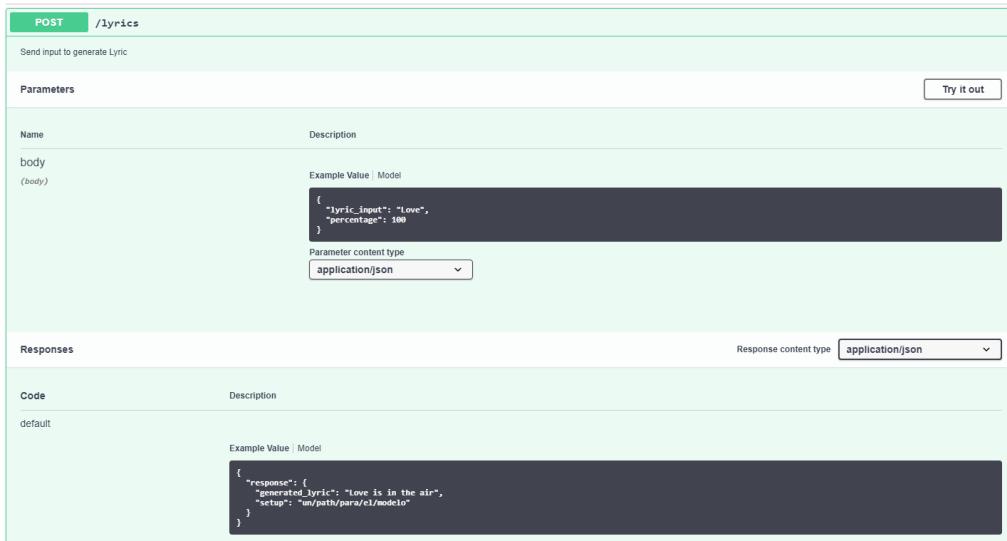


Figura 17: Cuadro del método Post

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        except Exception as e:
39            logging.error(f'Error processing {self.seed_text}: {str(e)}')
40        seed_text = self.seed_text
41
42    def complete_this_song(self, next_words, first_verse=False):
43        logging.info(f'Generating song.')
44        try:
45            seed_text = self.generate_rhyme(first_verse)
46            for _ in range(next_words):
47                token_list = self.tokenizer.texts_to_sequences([seed_text])
48                [0]
49                token_list = pad_sequences([token_list], maxlen=self.max_len
50                -1, padding='pre')
51                predicted = argmax(self.model.predict(token_list, verbose=0),
52                axis=-1)
53                output_word = ""
54                for word, index in self.tokenizer.word_index.items():
55                    if index == predicted:
56                        output_word = word
57                        break
58                seed_text += " " + output_word
59        except AttributeError:
60            seed_text = f'Cannot process {self.seed_text} try with a
different word.'
61            logging.error(f'Error processing {self.seed_text}:
AttributeError')
62
63        return seed_text, self.percentage

```

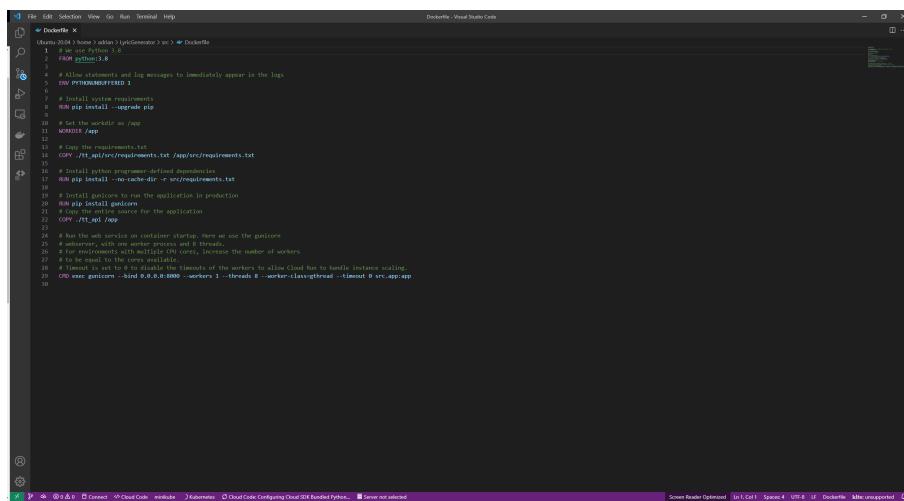
La palabra introducida por el usuario es enviada al modelo para ser procesada, generando de esta manera un texto con la letra de la canción, este procedimiento se realiza en la función “generate\_lyric” y el texto generado es regresado en formato json a la aplicación web en React, que será posteriormente mostrada en el navegador web del usuario.

## 15. Desplegando componentes

Una vez se tiene lista la comunicación con el modelo, se vuelve 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 establece la comunicación entre la página y la API de forma local se procede a desplegar estos componentes por separado ya que se manejan distintos lenguajes de programación y puede ocasionar error tener todo junto, así como por cuestiones de escalamiento y para evitar errores ocasionadas por las estructuras monolíticas.

### 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 que incluirá a la API y que pueda correr en cualquier sistema operativo compatible con Docker.



A screenshot of a Dockerfile in a Visual Studio Code editor. The code defines a Python application with the following commands:

```
File Edit Selection View Go Run Terminal Help
Dockerfile - Visual Studio Code
Ubuntu-20.04 home > addan > pygenGenerator > src > Dockerfile
1 FROM python:3.8
2 # Allow statements and log messages to immediately appear in the logs
3 ENV PYTHONUNBUFFERED=1
4 # Install system requirements
5 RUN apt-get update &amp; apt-get install -y curl
6 WORKDIR /app
7 COPY . /app
8 # Copy the requirements.txt
9 COPY ./flask_api/rec/requirements.txt ./app/src/requirements.txt
10 # Install python programme-defined dependencies
11 RUN pip install --no-cache-dir -r ./app/src/requirements.txt
12 # Copy the code to run the application in production
13 RUN pip install gunicorn
14 # Copy the entire source for the application
15 COPY ./flask_api /app
16 # If you are running in container, start your application
17 # with one worker process and 8 threads,
18 # otherwise, with one worker process and 8 threads.
19 # For environments with multiple IP cores, increase the number of workers
20 # to 10 or 12.
21 # Timeout is set to 60 to disable the timeout of the workers to allow Cloud Run to handle instance scaling.
22 # DO NOT put commas -&amp; 6.6.6.6:8000 -workers 1 -threads 8 -worker-class:gthread -timeout 60 src/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 en la plataforma <https://hub.docker.com/> con la finalidad de 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.

Una vez se cuenta con los 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 y así configurar las herramientas a conveniencia y según sus necesidades particulares), los pasos a seguir para poder crear un entorno en Elastic Beanstalk son los siguientes:

Se selecciona el tipo de entorno en el que se trabajará, en este caso es un entorno web ya que se trata de una API.

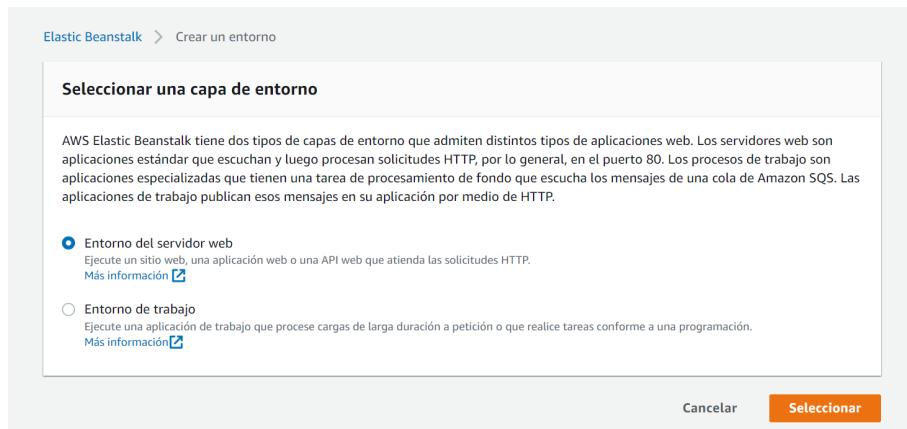


Figura 20: Tipos de entornos.

Se le da un nombre al entorno, se elige un nombre para generar el enlace 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 que se creará, en este caso se seleccionó como plataforma a Docker y se dejaron las configuraciones recomendadas por la misma plataforma.

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

Nombre del entorno	Neuralyrics-env
Campo	neuralyrics .us-east-1.elasticbeanstalk.
	<input type="button" value="Comprobar disponibilidad"/>
Descripción	Servicio para generar canciones llamando a un modelo generado.

### Plataforma

<input checked="" type="radio"/> Plataforma administrada Plataformas publicadas y mantenidas por Amazon Elastic Beanstalk. <a href="#">Obtenga más información</a>	<input type="radio"/> Plataforma personalizada Plataformas creadas por usted y que le pertenecen.
Plataforma	Docker
Ramificación de la plataforma	Docker running on 64bit Amazon Linux 2
Versión de la plataforma	3.4.9 (Recommended)

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

Como último paso requerido para crear el entorno es necesario cargar el código en un bucket alojado en Amazon S3, siendo este el archivo con extensión .YML creado previamente, y así 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 mostrar el siguiente estado cuando se encuentre listo:



Figura 24: Despliegue del entorno.

Si todo salió bien en este punto ya se puede vislumbrar la API dentro del link generado, sin embargo, hay algunas configuraciones del entorno general (Fig. 25) que requieren unos cambios para que no haya ningún inconveniente al conectar el Front-end por medio de SSL.

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 permitirá configurar después y de manera correcta los certificados para escuchar al puerto 443 de SSL.

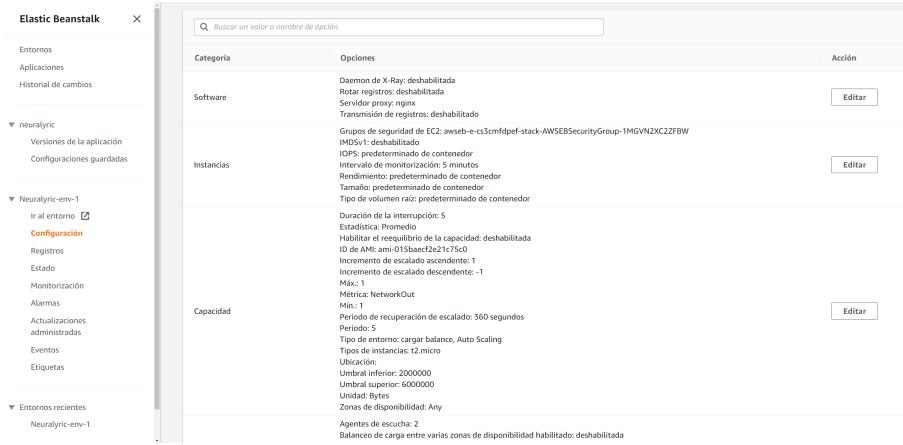


Figura 25: Primera parte de las configuraciones generales de Elastic Beans-talk.

Continuando con las configuraciones requeridas, 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.

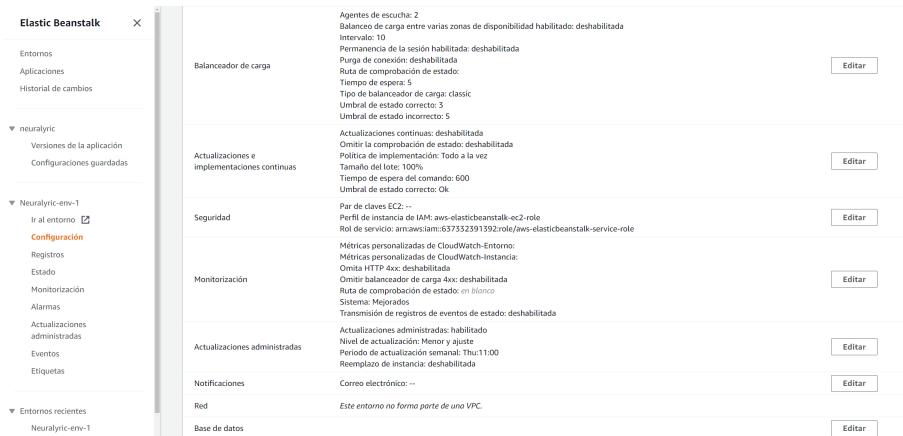


Figura 26: Segunda parte de las configuraciones generales de Elastic Beans-talk.

Elastic Beanstalk > Entornos > Neuralyric-env-1 > Configuración

### Modificar el balanceador de carga clásico

**Agentes de escucha**

Puede especificar agentes de escucha para el balanceador de carga. Cada agente de escucha dirige a las instancias el tráfico entrante desde el cliente 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.

	Puerto	Protocolo	Puerto de instancia	Protocolo de instancia	Certificado SSL	Habilit...
<input type="checkbox"/>	80	HTTP	80	HTTP	--	<input checked="" type="checkbox"/>
<input type="checkbox"/>	443	HTTPS	80	HTTP	*.neuralyrics.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 esté corriendo con los endpoints establecidos en el código de FLASK.

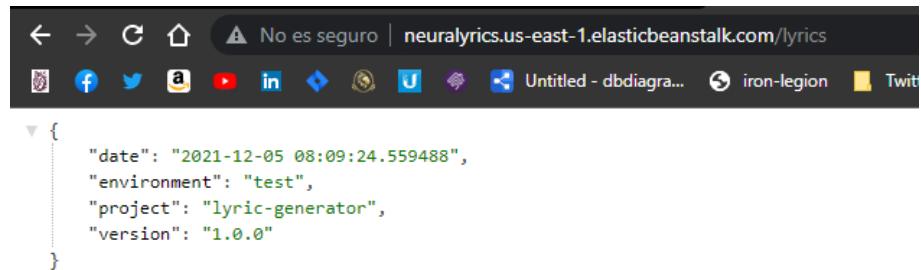


Figura 28: Estatus de la API corriendo con el endpoint configurado.

Una vez que la API se encuentre lista 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 la API pueda presentar en este punto puede afectar directamente a la página web, de igual forma se recomienda crear el certificado que aparece en la parte de “Configuración de SSL y DNS” para asegurar un correcto funcionamiento de la API usando 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 forma sencilla, además permite hacer despliegues de proyectos que se encuentren dentro de una cuenta de Github.

Lo primero que se requiere hacer para poder trabajar con la línea de comandos de Vercel es instalarlo, para ello debemos recurrir a nuestra terminal y descargarlo 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, se debe de abrir la carpeta de nuestro proyecto a desplegar y abrir una terminal, 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 preguntará quien lo va a desplegar, para ello usamos una cuenta creada en la plataforma de Vercel 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

Una vez se configuraron 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 nuestro 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 genera un enlace mediante el cual podemos ver el estado del despliegue, y de presentarse algún problema se nos estará indicando cual su origen en la misma terminal.

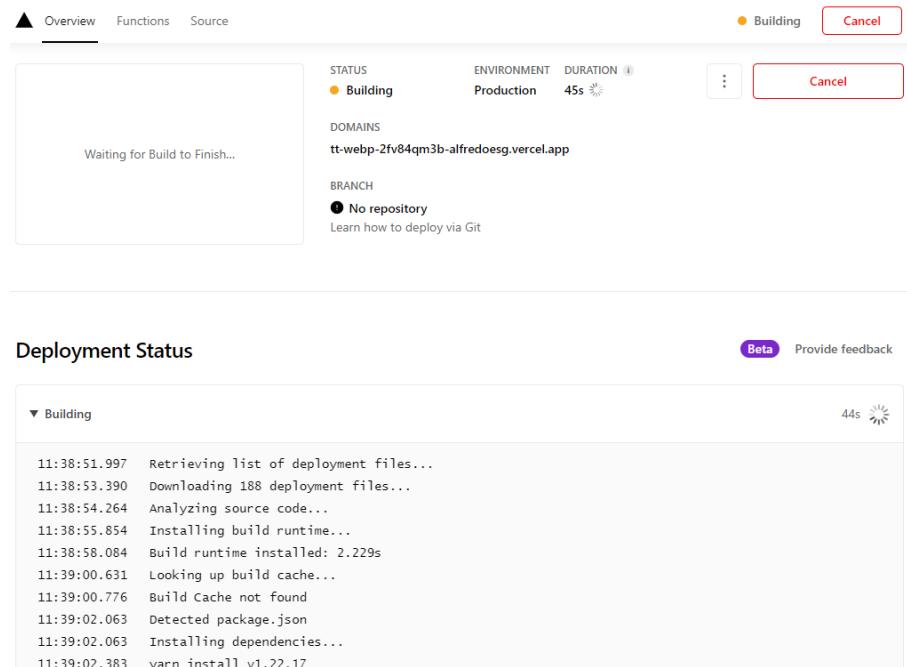


Figura 32: Desplegado el proyecto

Al momento de desplegar la aplicación nos marca un error.

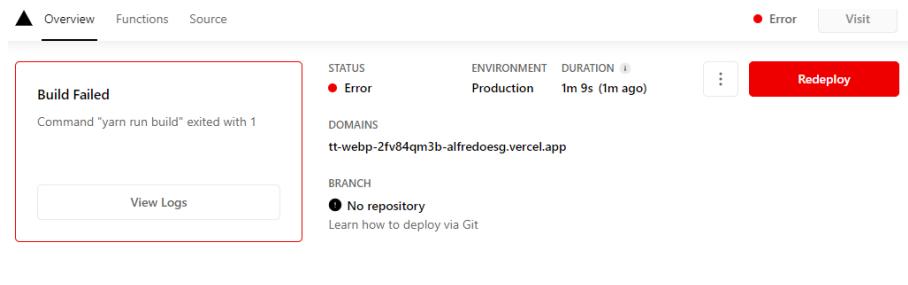


Figura 33: Desplegando el proyecto

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 por lo que para poder hacer el despliegue y que no nos marque error debemos entrar a las configuraciones de nuestro proyecto.

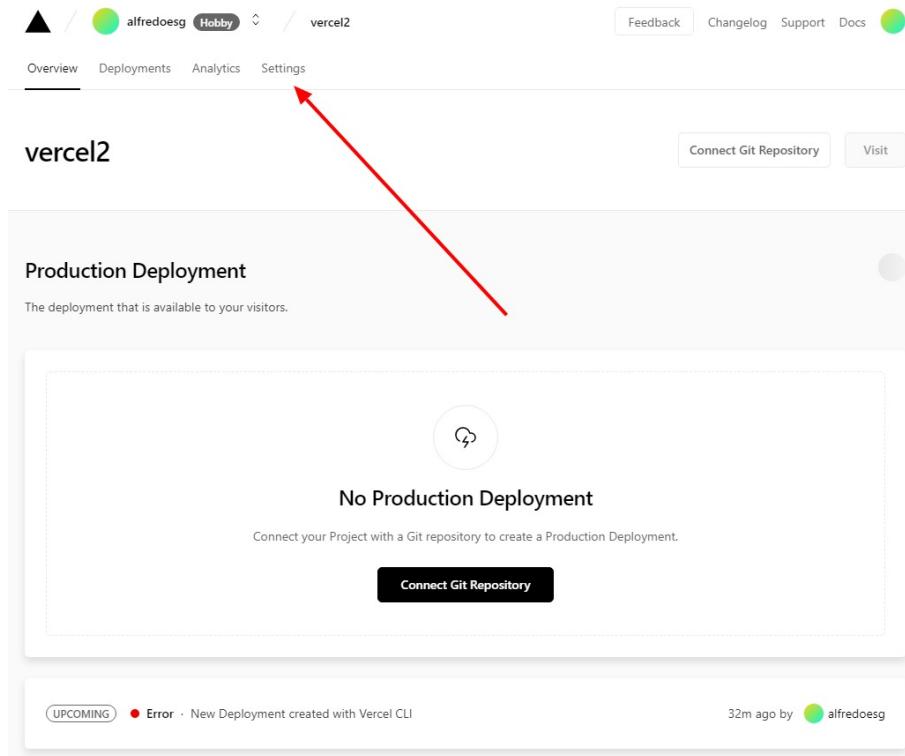


Figura 34: Configuración del proyecto desplegado

Una vez dentro de la configuración del proyecto se debe de ir a la sección de “variables de entorno” y ahí agregar la variable CI con un valor de false.

**Project Settings**

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

**Domains**      In order to provide your Deployment with Environment Variables at Build and Runtime, you may enter them right here, for the Environment of your choice.

**Integrations**

**Git**      A new Deployment is required for your changes to take effect.

**Serverless Functions**

**Environment Variables**

**Security**

**Advanced**

**Add New**

NAME	ENVIRONMENT
CI	<input checked="" type="checkbox"/> Production
	<input checked="" type="checkbox"/> Preview <a href="#">Select Custom Branch</a>
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

Habiendo realizado esos cambios se regresa a la terminal y se vuelve a escribir el comando “vercel” y automáticamente volverá a realizar el despliegue de la aplicación web. Con el comando vercel escrito en la terminal también es posible actualizar la página desplegada en caso de haber algún cambio en el código.

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

**Overview**      **Functions**      **Source**

**Neural Lyrics**

**STATUS** Ready      **ENVIRONMENT** Preview      **DURATION** 1m 15s (1m ago)      **⋮**      **Visit**

**DOMAINS**  
tt-webs-j6b1b7ika-alfredoegs.vercel.app

**BRANCH**  
● No repository  
Learn how to deploy via Git

Figura 36: Aplicación web desplegada

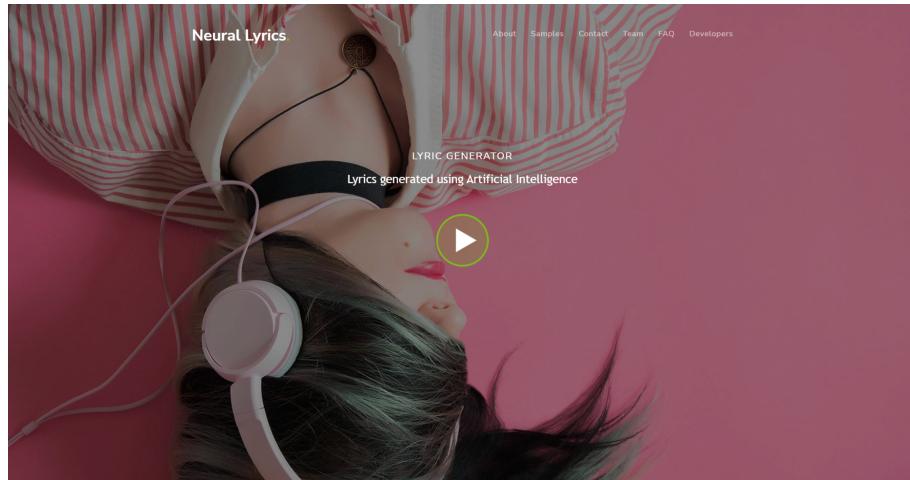


Figura 37: Accediendo a la aplicación web desplegada

En nuestro caso, previamente se había adquirido un dominio por lo que para realizar el cambio del enlace proporcionado por Vercel a el enlace del dominio adquirido hay que acceder nuevamente a la configuración del proyecto y ahí en la sección de “dominio” se agrega la nueva dirección desde la que queremos acceder.

## 16. Configuración de SSL y DNS

Para permitir que nuestra página web pueda accederse desde otros dispositivos es necesario hacer la configuración correcta de DNS y proporcionarle un certificado de seguridad. Dado que nuestra página web fue desplegada en la plataforma de Vercel, para que esta pueda direccionar de manera correcta usando nuestro dominio personal se requiere de la configuración de un DNS específico requerido por 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 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 de manera 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	<a href="#">Editar</a>
CNAME	api	neuralyrics.us-east-1.elasticbean...	Cloudflare Solo DNS	Automático	<a href="#">Editar</a>
CNAME	_b78ca08e69d8db508e734a...	_495f1a7b8eb5337bb83d7fc34...	Cloudflare Solo DNS	Automático	<a href="#">Editar</a>
CNAME	www	cname.vercel-dns.com	Cloudflare Solo DNS	Automático	<a href="#">Editar</a>

Figura 41: Configuración final de los registros DNS.

1. El primero es la dirección de DNS proporcionado por Vercel de tipo A, el cual permite enlazar una dirección con un nombre de dominio.
2. El segundo es el código del certificado creado en AWS de tipo CNAME con el nombre y contenido proporcionado por AWS.
3. El tercero es otro tipo CNAME con el nombre de “api” y conteniendo a la API directamente.
4. El cuarto es el valor de tipo CNAME proporcionado por Vercel.

Una vez desplegadas la página web y la API, y si todo se hizo correctamente, deberá ser posible comunicarse mediante los links generados al final de cada sección, cada parte tendrá su propio versionamiento, y una ventaja de esto es que pueden crecer los dos servicios de forma separada sin necesidad de depender de ellos, con lo que si una incurre en un error no deberá afectar el acceso al enlace 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](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](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]