



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
0.1. Presentación	6
0.2. Resumen	6
0.3. Introducción	7
0.4. Objetivo	7
0.5. Requerimientos mínimos técnicos	8
0.6. Requerimientos mínimos de software	8
0.7. Herramientas utilizadas para el desarrollo	9
0.7.1. Python	9
0.7.2. HTML	9
0.7.3. CSS	9
0.7.4. JavaScript	10
0.7.5. Flask	10
0.7.6. Gunicorn	11
0.7.7. Amazon EC2	11
0.7.8. Amazon SageMaker	11
0.7.9. Amazon S3	12
0.8. Diagramas de la aplicación web	13
0.8.1. Diagrama de casos de uso	13
0.9. Funcionamiento general del sistema	15
0.10. Base de datos	16
0.11. Limpieza de la base de datos	18
0.12. Desarrollo del modelo	20
0.13. Creación del modelo	23
0.13.1. Embedding o incrustación	23
0.13.2. Bidireccional	23
0.13.3. Dropout	23
0.13.4. Densidad	24
0.13.5. Método de compilación, algoritmo de optimización y métrica de rendimiento	24

0.13.6. Importación del modelo	25
0.14. Desarrollo de la aplicación web	26
0.15. Desarrollo del back-end	33
0.16. Conexión entre el modelo y la aplicación web	37

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 datos	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 Back-end	35
17.	Cuadro del método Post	36

Índice de cuadros.

1.	Caso de uso 1	14
2.	Caso de uso 2	14
3.	Caso de uso 3	14
4.	Caso de uso 4	14

0.1. 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.

0.2. 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.

0.3. 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.

0.4. 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.

0.5. Requerimientos mínimos técnicos

- Procesador: Intel Core i3
- Memoria RAM: 4 Gb
- Disco duro: 500 Gb

0.6. Requerimientos mínimos de software

- Sistema Operativo: Windows 7/8.1/10

0.7. Herramientas utilizadas para el desarrollo

0.7.1. 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]

0.7.2. 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 APIs y un mejor diseño usando CSS3. [3]

0.7.3. 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.

0.7.4. 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.

0.7.5. 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 describe 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.

0.7.6. Unicorn

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 implemente la especificación WSGI de frameworks como Django, Flask o Bottle.

0.7.7. 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.

0.7.8. Amazon SageMaker

Amazon SageMaker [7] es un servicio que ayuda a científicos y desarrolladores a construir, entrenar e implementar de manera rápida y sencilla modelos de machine learning.

Para construir el modelo, este servicio cuenta con algoritmos de machine learning más utilizados que vienen preinstalados. También está preconfigurado para que pueda ejecutar Apache MXNet y TensorFlow.

Para el entrenamiento, con un solo clic en la consola de servicio, es fácil comenzar a entrenar su modelo. Amazon SageMaker se encarga de cada infraestructura y facilita la escalabilidad. lo que permite entrenar los modelos a escala de peta-bytes Si se desea acelerar y simplificar el proceso de entrenamiento, se puede ajustar automáticamente el modelo para obtener la mejor precisión.

Para desplegar el modelo entrenado, se aloja en un clúster de escalado automático de Amazon EC2.

0.7.9. 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.

0.8. Diagramas de la aplicación web

0.8.1. 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.

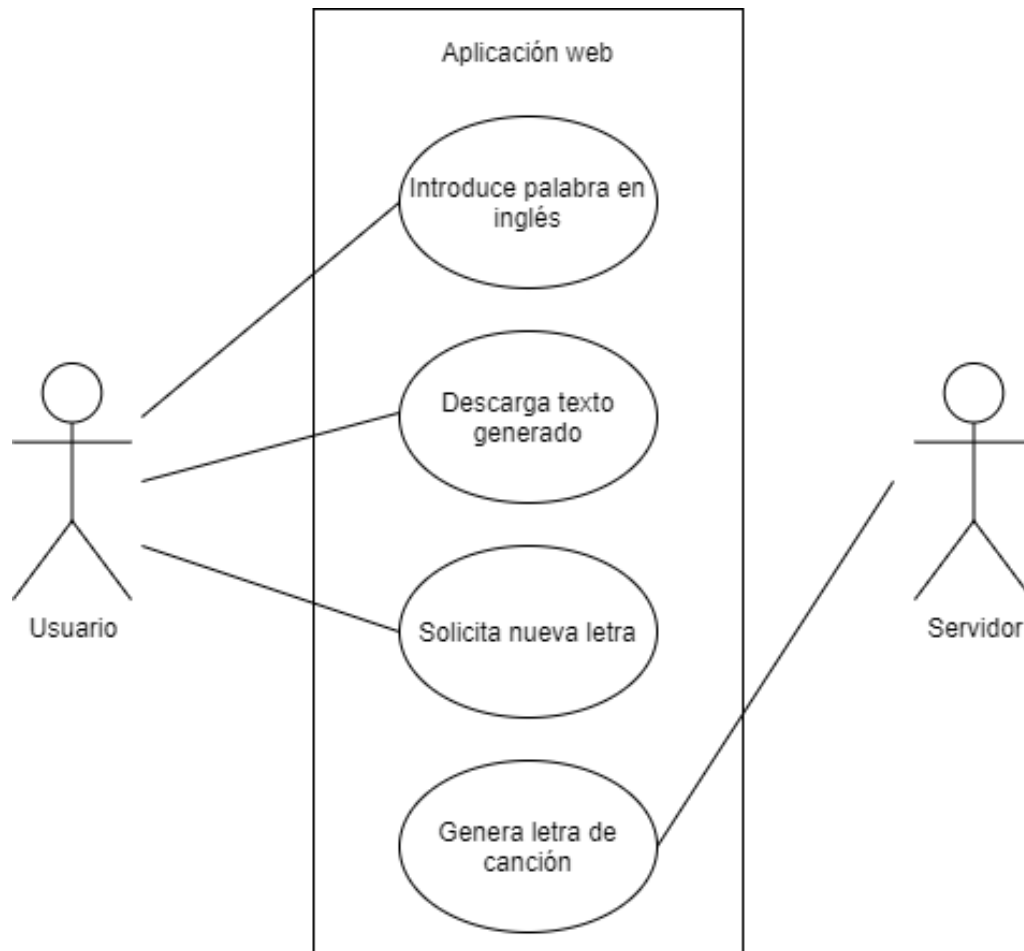


Figura 1: Diagrama de casos de uso

Cuadro 1: 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 2: 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 3: 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 4: 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

0.9. Funcionamiento general del sistema

El siguiente diagrama muestra el funcionamiento general del producto, donde se ejemplifica 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.

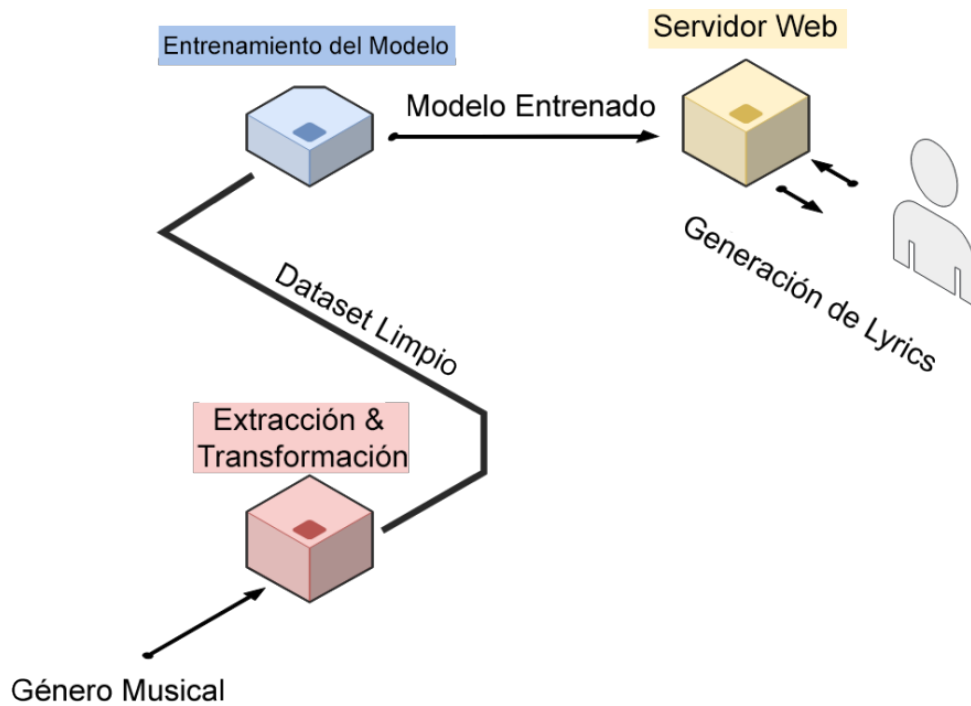


Figura 2: Diagrama general del sistema

0.10. 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:

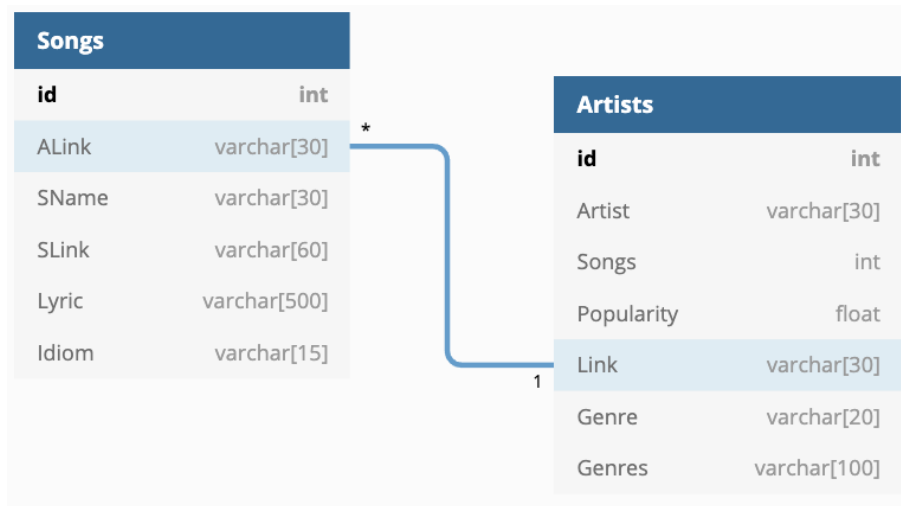


Figura 3: Diagrama entidad-relación de la base de datos

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

0.11. 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 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

0.12. Desarrollo del modelo

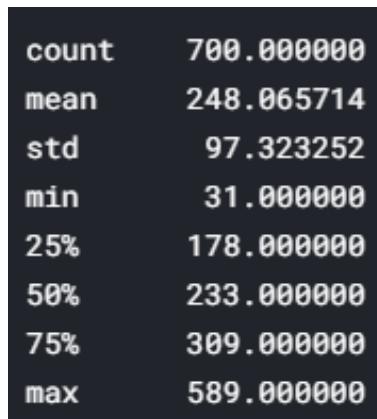
Para el desarrollo del modelo se utilizó una libreta de Kaggle. Cabe mencionar que el lenguaje de programación utilizado para el desarrollo del modelo fue Python, lo primero que se trabajó en la libreta, fue realizar la importación de librerías que se fueran a necesitar, las más importantes son:

- 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 numero 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 realizó 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.
- `_text_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.

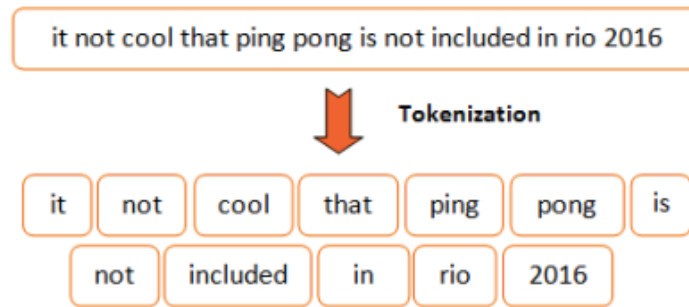


Figura 8: Tokenizado de las palabras [13]

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.

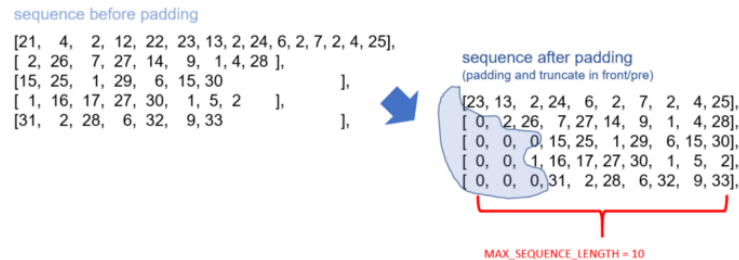


Figura 9: Padding del tokenizado de las palabras

0.13. 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:

0.13.1. 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))
```

0.13.2. 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)))
```

0.13.3. 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 sobreajuste.


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

0.13.4. 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'))
```

0.13.5. 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])

```

0.13.6. 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,
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

```

0.14. 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 pueden 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 ejecutará 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.

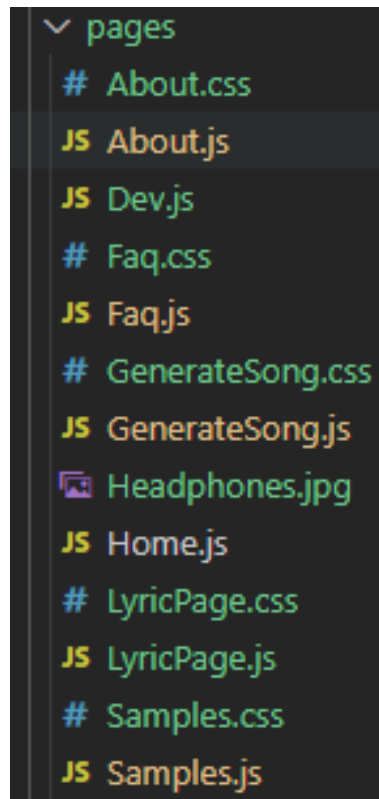


Figura 10: Páginas trabajadas

La más importante de estas páginas es la página principal, debido a que es esta la primera 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.

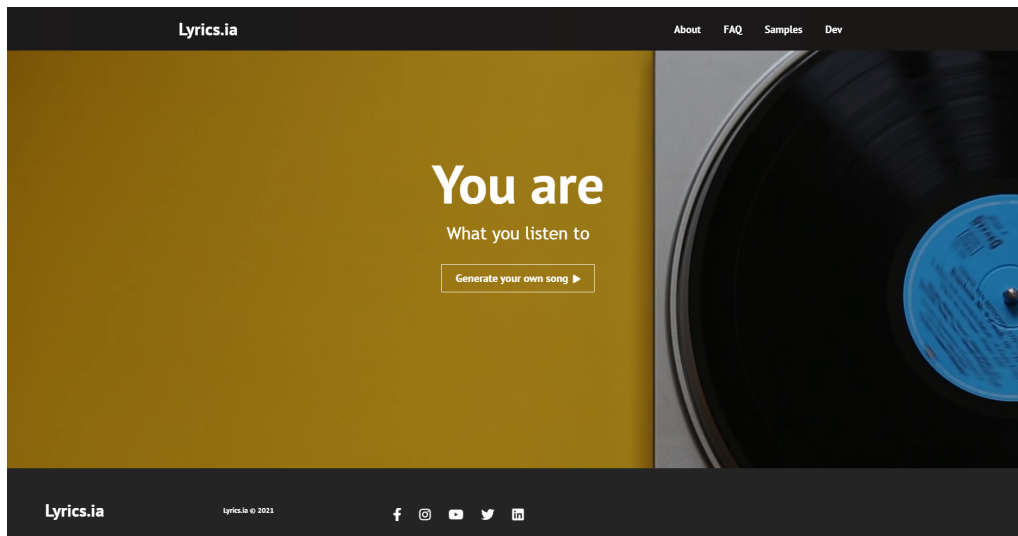


Figura 11: Página de bienvenida

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 back-end 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('http://localhost:5000/lyrics', {
7         method: 'POST',
8         headers:{'content-type':'application/json','Access-
9 Control-Allow-Origin':'*'},
10        body:JSON.stringify({"lyric_input":engword,"
11 percentage":percentval})
12    }).then(response => {
13        return response.json()
14    }).then(json => {
15        this.setState({englishword: json[0]})
16    }).catch(error => {
17        console.log(error)
18    })
19 }
```

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 back-end usando la función de fetch, lo que hace dicha función es buscar la url, que en este caso es donde se encuentra alojado nuestro back-end y busca el método post de esta, como el método post de nuestro back-end 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.

Lyric generada

Nuestros gatos compositores estan trabajando

Descargar

[Generar nuevamente](#)

[Nuevos parametros](#)

Figura 13: Leyenda mostrada

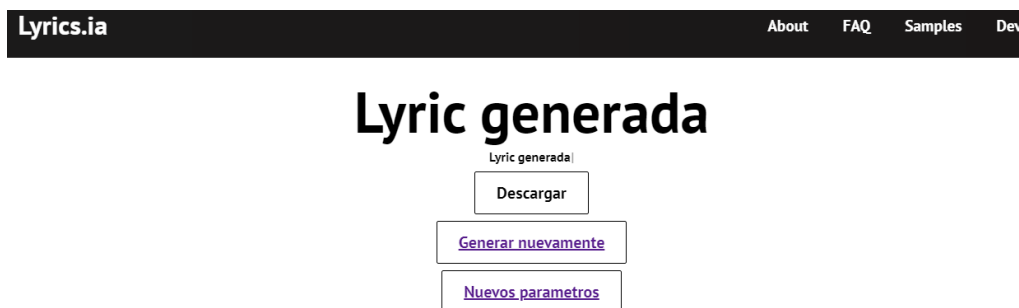


Figura 14: Texto generado

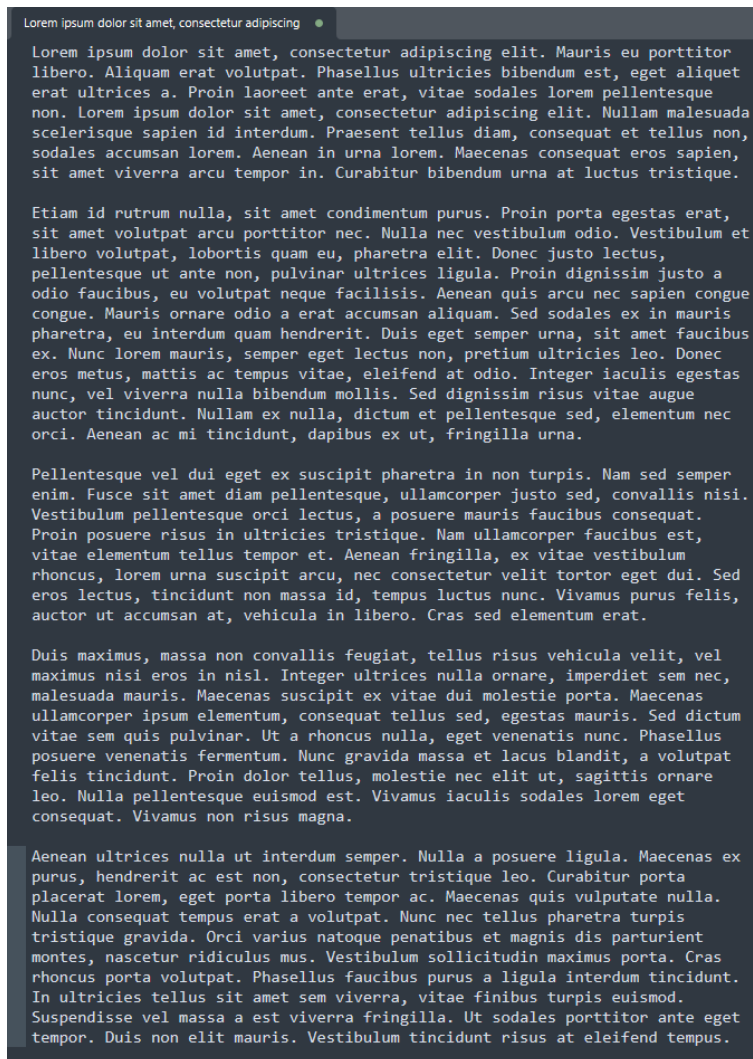
Adentrándonos a esta sección, la primera función que debemos revisar es la que permite hacer la conexión con el back-end para que la página pueda recibir el texto generado por el modelo.

```
1   useEffect(() => {
2       fetchLyricAsync ()
3   }, [])
4
5   const fetchLyricAsync = async () =>{
6       const result = await axios.get("")
7       console.log(result)
8   }
```

Lo que se hace en el código anterior es realizar una conexión asíncrona con el back-end, 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.



>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris eu porttitor libero. Aliquam erat volutpat. Phasellus ultricies bibendum est, eget aliquet erat ultrices a. Proin laoreet ante erat, vitae sodales lorem pellentesque non. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam malesuada scelerisque sapien id interdum. Praesent tellus diam, consequat et tellus non, sodales accumsan lorem. Aenean in urna lorem. Maecenas consequat eros sapien, sit amet viverra arcu tempor in. Curabitur bibendum urna at luctus tristique.

Etiam id rutrum nulla, sit amet condimentum purus. Proin porta egestas erat, sit amet volutpat arcu porttitor nec. Nulla nec vestibulum odio. Vestibulum et libero volutpat, lobortis quam eu, pharetra elit. Donec justo lectus, pellentesque ut ante non, pulvinar ultrices ligula. Proin dignissim justo a odio faucibus, eu volutpat neque facilisis. Aenean quis arcu nec sapien congue congue. Mauris ornare odio a erat accumsan aliquam. Sed sodales ex in mauris pharetra, eu interdum quam hendrerit. Duis eget semper urna, sit amet faucibus ex. Nunc lorem mauris, semper eget lectus non, pretium ultricies leo. Donec eros metus, mattis ac tempus vitae, eleifend at odio. Integer iaculis egestas nunc, vel viverra nulla bibendum mollis. Sed dignissim risus vitae augue auctor tincidunt. Nullam ex nulla, dictum et pellentesque sed, elementum nec orci. Aenean ac mi tincidunt, dapibus ex ut, fringilla urna.

Pellentesque vel dui eget ex suscipit pharetra in non turpis. Nam sed semper enim. Fusce sit amet diam pellentesque, ullamcorper justo sed, convallis nisi. Vestibulum pellentesque orci lectus, a posuere mauris faucibus consequat. Proin posuere risus in ultricies tristique. Nam ullamcorper faucibus est, vitae elementum tellus tempor et. Aenean fringilla, ex vitae vestibulum rhoncus, lorem urna suscipit arcu, nec consectetur velit tortor eget dui. Sed eros lectus, tincidunt non massa id, tempus luctus nunc. Vivamus purus felis, auctor ut accumsan at, vehicula in libero. Cras sed elementum erat.

Duis maximus, massa non convallis feugiat, tellus risus vehicula velit, vel maximus nisi eros in nisl. Integer ultrices nulla ornare, imperdiet sem nec, malesuada mauris. Maecenas suscipit ex vitae dui molestie porta. Maecenas ullamcorper ipsum elementum, consequat tellus sed, egestas mauris. Sed dictum vitae sem quis pulvinar. Ut a rhoncus nulla, eget venenatis nunc. Phasellus posuere venenatis fermentum. Nunc gravida massa et lacus blandit, a volutpat felis tincidunt. Proin dolor tellus, molestie nec elit ut, sagittis ornare leo. Nulla pellentesque euismod est. Vivamus iaculis sodales lorem eget consequat. Vivamus non risus magna.

Aenean ultrices nulla ut interdum semper. Nulla a posuere ligula. Maecenas ex purus, hendrerit ac est non, consectetur tristique leo. Curabitur porta placerat lorem, eget porta libero tempor ac. Maecenas quis vulputate nulla. Nulla consequat tempus erat a volutpat. Nunc nec tellus pharetra turpis tristique gravida. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Vestibulum sollicitudin maximus porta. Cras rhoncus porta volutpat. Phasellus faucibus purus a ligula interdum tincidunt. In ultricies tellus sit amet sem viverra, vitae finibus turpis euismod. Suspendisse vel massa a est viverra fringilla. Ut sodales porttitor ante eget tempor. Duis non elit mauris. Vestibulum tincidunt risus at eleifend tempus.

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 back-end 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.

0.15. Desarrollo del back-end

Para el back-end 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  from datetime import datetime
3  from flask import Flask, jsonify
4  from flask_restful import Resource, Api
5  from flask_apispec import marshal_with, doc, use_kwargs
6  from flask_apispec.views import MethodResource
7  from apispec import APISpec
8  from apispec.ext.marshmallow import MarshmallowPlugin
9  from flask_apispec.extension import FlaskApiSpec
10 from flask_cors import CORS
11
12 # Own Libraries
13 from utils.schemas import HealthSchema, GetLyrics, PostLyrics
14 from utils.generate_lyrics import GenerateLyric
15 from utils.constants import FLASK_ENV, VERSION, PROJECT
16
17 app = Flask(__name__)
18 api = Api(app) # Flask restful wraps Flask app around it.
19 cors = CORS(app)
20
21 app.config.update({
22     'APISPEC_SPEC': APISpec(
23         title='Lyric Generator',
24         version='v1',
25         plugins=[MarshmallowPlugin()],
26         openapi_version='2.0',
27     ),
28     'APISPEC_SWAGGER_URL': '/swagger/', # URI to access API Doc JSON
29     'APISPEC_SWAGGER_UI_URL': '/swagger-ui/' # URI to access UI of API Doc
30 })
31 docs = FlaskApiSpec(app)
32
33 # Endpoints
34 class Lyrics(MethodResource, Resource):
35     """
36     Source endpoint to check health of the API
37     """
38     @doc(description='Health-check to see the status of the API.', tags=['
39     Lyric Generator Status'])
40     @marshal_with(HealthSchema)
41     def get(self):
42         """
43         Get method represents a GET API method
```

```

43         """
44         return {
45             'project': PROJECT,
46             'version': VERSION,
47             'environment': FLASK_ENV,
48             'date': datetime.now(),
49         }
50
51     """
52     Class to generate the lyric of the API
53     """
54     @doc(description='Send input to generate Lyric', tags=['Generate Lyric',
55 ])
56     @use_kwargs(GetLyrics, location=('json'))
57     @marshal_with(PostLyrics)
58     def post(self, **kwargs):
59         """
60         Post method represents a POST API method
61         """
62         try:
63             response = {}
64             lyrics = GenerateLyric(**kwargs)
65             response["setup"] = lyrics.setup()
66             response["generated_lyric"] = lyrics.generate_lyric()
67         except Exception as e:
68             return {"error": f'Error:{str(e)}'}
69
70         return {"response": response}
71
72     @doc(description='Health-check to see the status of the API.', tags=['
73 Lyric Generator Status'])
74     @marshal_with(HealthSchema)
75     def get(self):
76         """
77         Get method represents a GET API method
78         """
79         return {
80             'project': PROJECT,
81             'version': VERSION,
82             'environment': FLASK_ENV,
83             'date': datetime.now(),
84         }
85
86 # Add Endpoints
87 api.add_resource(Lyrics, '/lyrics')
88
89 # Add Swagger Documentation
90 docs.register(Lyrics)
91
92 # Handling of 404 errors.
93 @app.errorhandler(404)
94 def page_not_found(e):
95     return jsonify({"error": "resource not found", "code": "404"}), 404
96
97 # We run the Flask application and, if it is running in a development
98 # environment,
99 # we run the application in debug mode.
100 app.run(debug=FLASK_ENV == 'development', host='0.0.0.0')

```

En el código app.py mostrado anteriormente es el código principal de nuestro

back-end 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 back-end se realizó una pequeña interfaz y es posible acceder a ella a través de la siguiente dirección `x/swagger-ui/` en ella podemos encontrar con mayor detalle el status actual de la api, la cual se encuentra ubicada en el espacio nombrado como `health`.

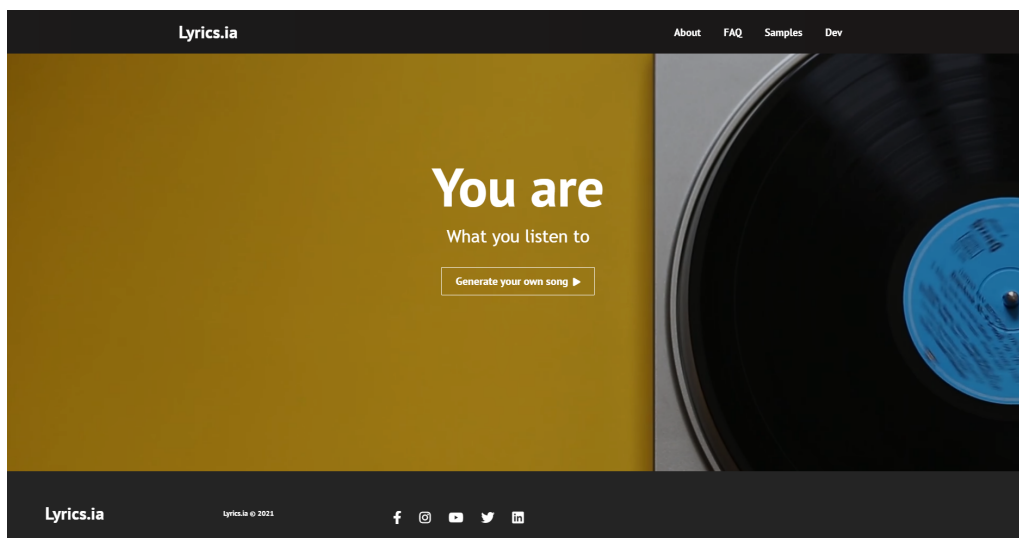


Figura 16: Cuadro de status del Back-end

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

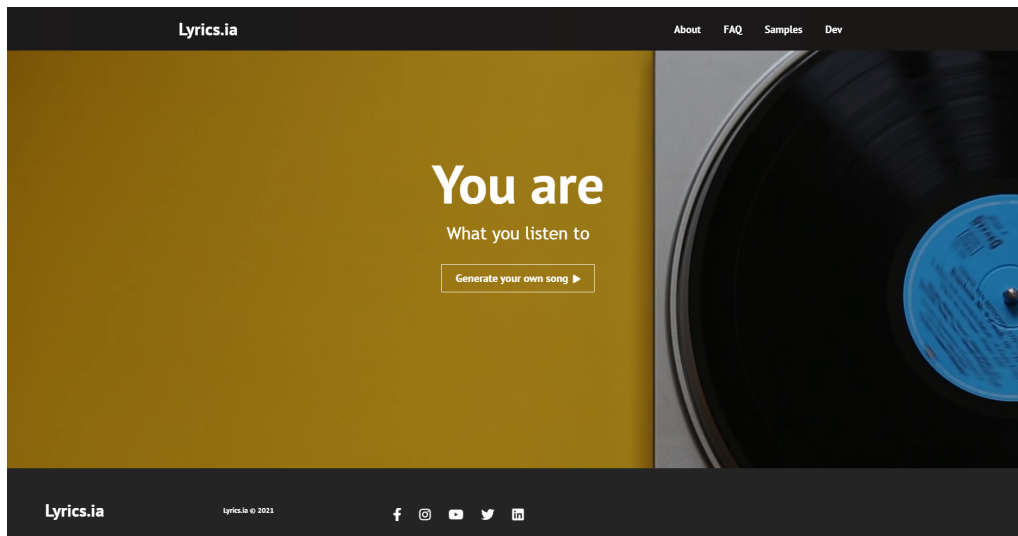


Figura 17: Cuadro del método Post

Para realizar esta tarea se desarrolló el siguiente código

```
1 from utils.constants import MODEL_PATH
2
3 class GenerateLyric(object):
4     """
5     TODO:
6     - Generate Model and integrate it in this class.
7     - Integrate it in React
8     """
9     def setup(self):
10         return MODEL_PATH
11
12     def generate_lyric(self, lyric_input, percentage):
13         return lyric_input + " Is in the air", percentage
14
15     def show(self):
16         response = "Hola soy una lyric"
17         return response
```

En este código utilizando la palabra y el porcentaje introducidos por el usuario son enviados al modelo para ser procesados generando así un texto, este procedimiento se realiza en la función `generate_lyric`, e; texto generado va a ser regresado a nuestra api para mandarlo a la aplicación web en React para posteriormente mostrarla en el navegador web del usuario.

0.16. Conexión entre el modelo y la aplicación web

Bibliografía

- [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] Amazon (2021), Amazon SageMaker [En línea]. Disponible: <https://aws.amazon.com/es/sagemaker/> [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]