

# Manual Técnico

Traductor Mongo DB

Fecha: 30 de abril de 2024

## Contenido

---

OBJETIVO .....	3
ALCANCE .....	3
DESARROLLO DEL MANUAL TÉCNICO .....	4
Descripción General .....	4
Requisitos del Sistema .....	4
Instalación .....	4
Descripción del funcionamiento .....	4
Interfaz Gráfica .....	4
Análisis Léxico .....	6
Analizador Sintáctico .....	6
Tabla de tokens .....	7
Diagrama de Árbol .....	8
Autómata Finito Determinista .....	8
Conjunto de Estados (N) .....	8
Alfabeto de Entrada ( $\Sigma$ ) .....	9
Estado Inicial ( $S_0$ ): .....	9
Función de Transición .....	9
Gramática Libre de Contexto .....	10
Traducción a MongoDB .....	11
Generación de Resultados .....	12
Estructura del proyecto .....	13
Manejo de Errores .....	14
Áreas de Mejoras Futuras .....	14

## OBJETIVO

---

El objetivo de este manual técnico es proporcionar a cualquier persona interesada en el proyecto una guía completa sobre la estructura, funcionamiento y desarrollo del traductor de sentencias MongoDB. Este manual tiene como propósito facilitar la comprensión de cómo se ha implementado la aplicación, permitiendo su mantenimiento, extensión y personalización de acuerdo con las necesidades cambiantes del usuario.

## ALCANCE

---

Traductor MongoDB – Este manual abarca aspectos como la descripción, propósito y funciones disponibles del proyecto, características principales, requisitos mínimos del sistema, instalación y ejecución del mismo, incluyendo cualquier configuración adicional que pueda ser necesaria.

# DESARROLLO DEL MANUAL TÉCNICO

## Descripción General

---

El Traductor MongoDB es una aplicación desarrollada en Python con interfaz gráfica utilizando la biblioteca Tkinter. Su objetivo principal es analizar un texto mediante un archivo de entrada, el cual tiene un formato específico y traducirlo a sentencias aptas para MongoDB, a su vez, se genera un reporte con los lexemas y tokens obtenidos mediante un analizador léxico y un sintáctico o bien si el archivo posee errores o caracteres inválidos también se genera un reporte de ellos.

## Requisitos del Sistema

---

- Python 3.x instalado en el sistema.
- Bibliotecas necesarias instaladas: tkinter.

## Instalación

---

Debido a que es una aplicación que se ejecuta con interfaz gráfica, se debe clonar o descargar el repositorio del proyecto desde Github para luego abrir el proyecto en el IDE de su preferencia (se recomienda Visual Studio Code) y correr la aplicación.

## Descripción del funcionamiento

### Interfaz Gráfica

---

La aplicación utiliza Tkinter para proporcionar una interfaz gráfica simple al usuario. La interfaz incluye:

- Un menú Archivo con las siguientes opciones:
  - Nuevo
  - Abrir
  - Guardar
  - Guardar como

- Salir
- Un menú Análisis con la opción “Traducir a MongoDB”
- Menú Token y Menú Errores
- Dos áreas de texto: una para ingresar el texto a traducir y otra para mostrar el resultado en sentencias MongoDB.

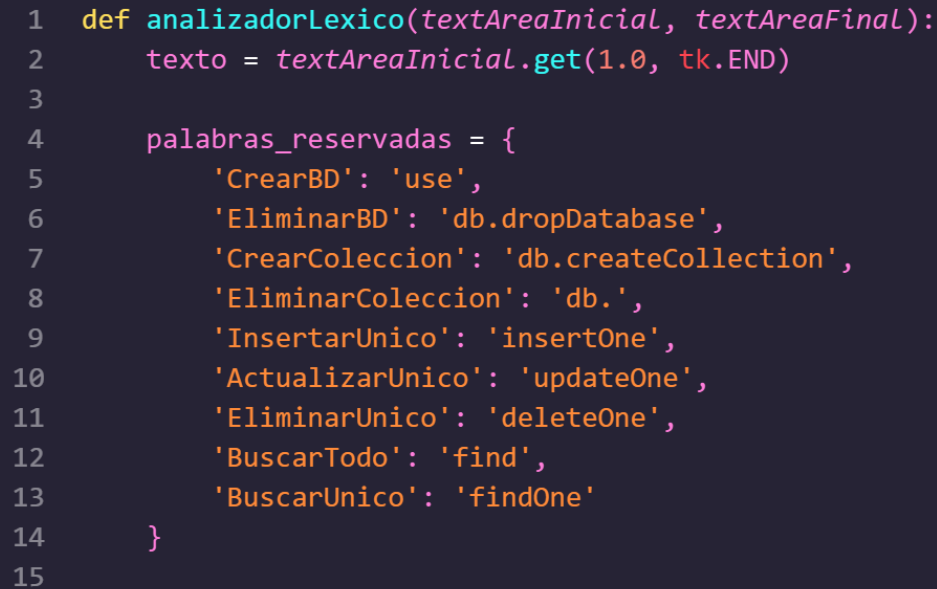
```

1 # Crear la ventana principal
2 ventana = tk.Tk()
3 ventana.title("Traductor Mongo DB")
4 ventana.configure(bg="#9195F6")
5 ventana.geometry("900x600")
6
7 # Menú de opciones
8 menu_opciones = tk.Menu(ventana)
9 ventana.config(menu=menu_opciones)
10
11 # Opción Archivo
12 submenu_archivo = tk.Menu(menu_opciones, tearoff=0)
13 menu_opciones.add_cascade(label="Archivo", menu=submenu_archivo)
14 submenu_archivo.add_command(label="Nuevo", command=nuevo)
15 submenu_archivo.add_separator()
16 submenu_archivo.add_command(label="Abrir ", command=abrir_archivo)
17 submenu_archivo.add_separator()
18 submenu_archivo.add_command(label="Guardar", command=guardar)
19 submenu_archivo.add_separator()
20 submenu_archivo.add_command(label="Guardar como", command=guardar_como)
21 submenu_archivo.add_separator()
22 submenu_archivo.add_command(label="Salir", command=salir)
23
24 # Opción Análisis
25 submenu_analisis = tk.Menu(menu_opciones, tearoff=0)
26 menu_opciones.add_cascade(label="Análisis", menu=submenu_analisis)
27 submenu_analisis.add_command(label="Traducir a Mongo DB", command=lambda: insertar_sentencias(textAreaInicial, textAreaFinal))
28
29 # Opción Tokens
30 submenu_tokens = tk.Menu(menu_opciones, tearoff=0)
31 menu_opciones.add_cascade(label="Tokens", menu=submenu_tokens)
32 submenu_tokens.add_command(label="Generar tabla", command=lambda: generar_tabla_tokens(lexemas))
33
34 # Opción Errores
35 submenuErrores = tk.Menu(menu_opciones, tearoff=0)
36 menu_opciones.add_cascade(label="Errores", menu=submenuErrores)
37
38 # Contenedor para las TextAreas
39 frame_textareas = tk.Frame(ventana, bg="#9195F6")
40 frame_textareas.pack(expand=True, fill=tk.BOTH, padx=20, pady=20)
41
42 # Primer textarea
43 textAreaInicial = tk.Text(frame_textareas, height=20, width=40)
44 textAreaInicial.pack(side=tk.LEFT, padx=10, pady=10, expand=True, fill=tk.BOTH)
45
46 # Segundo textarea
47 textAreaFinal = tk.Text(frame_textareas, height=20, width=40, state="normal")
48 textAreaFinal.pack(side=tk.LEFT, padx=10, pady=10, expand=True, fill=tk.BOTH)
49
50 ventana.mainloop()

```

## Análisis Léxico

Este componente se encarga de analizar el código fuente ingresado y convertirlo en una secuencia de tokens (lexemas). Se identifican tokens con sus valores correspondientes, como:

A screenshot of a code editor with a dark background and light-colored text. The code is a Python function named `analizadorLexico` that takes two parameters: `textAreaInicial` and `textAreaFinal`. It retrieves text from `textAreaInicial` and defines a dictionary of reserved words. The dictionary maps UI actions to database commands. The code is numbered from 1 to 15.

```
1 def analizadorLexico(textAreaInicial, textAreaFinal):
2     texto = textAreaInicial.get(1.0, tk.END)
3
4     palabras_reservadas = {
5         'CrearBD': 'use',
6         'EliminarBD': 'db.dropDatabase',
7         'CrearColeccion': 'db.createCollection',
8         'EliminarColeccion': 'db.',
9         'InsertarUnico': 'insertOne',
10        'ActualizarUnico': 'updateOne',
11        'EliminarUnico': 'deleteOne',
12        'BuscarTodo': 'find',
13        'BuscarUnico': 'findOne'
14    }
15
```

## Analizador Sintáctico

El analizador sintáctico interpreta la secuencia de tokens generada por el analizador léxico y realiza el análisis sintáctico para determinar la estructura y la semántica del código.

```

1  ef analizadorSintactico(Lexemas, errores):
2      sentencias_generadas = []
3
4      idx = 0
5      while idx < len(Lexemas):
6          idx += 1
7          if Lexemas[idx].tipo == 'Palabra Reservada':
8              if Lexemas[idx].lexema == 'use':
9                  idx = analizar_crear_bd(Lexemas, idx, sentencias_generadas, errores)
10
11             elif Lexemas[idx].lexema == 'dropDatabase':
12                 idx = analizar_eliminar_bd(Lexemas, idx, sentencias_generadas, errores)
13             elif Lexemas[idx].lexema == 'createCollection':
14                 idx = analizar_crear_coleccion(Lexemas, idx, sentencias_generadas, errores)
15             elif Lexemas[idx].lexema == 'dropCollection':
16                 idx = analizar_eliminar_coleccion(Lexemas, idx, sentencias_generadas, errores)
17             elif Lexemas[idx].lexema == 'find':
18                 idx = analizar_buscar_todo(Lexemas, idx, sentencias_generadas, errores)
19             elif Lexemas[idx].lexema == 'findOne':
20                 idx = analizar_buscar_unico(Lexemas, idx, sentencias_generadas, errores)
21             else:
22                 errores.append(Error(Lexemas[idx].lexema, Lexemas[idx].fila, Lexemas[idx].columna))
23                 idx += 1
24         else:
25             errores.append(Error(Lexemas[idx].lexema, Lexemas[idx].fila, Lexemas[idx].columna))
26             idx += 1
27
28     return sentencias_generadas

```

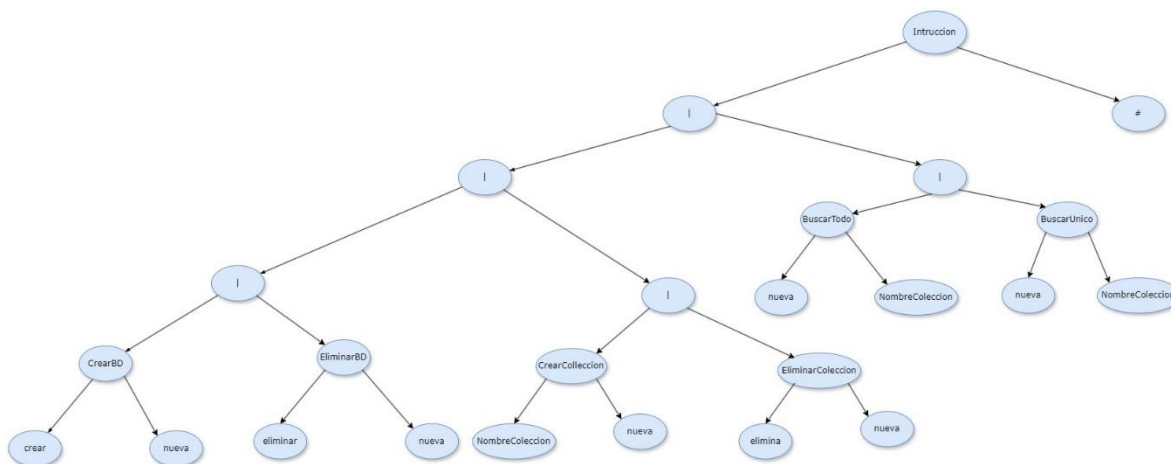
## Tabla de tokens

La siguiente tabla proporciona una lista de los tokens reconocidos por el Analizador Léxico del traductor MongoDB, junto con el patrón de expresión regular que define cada token:

Token	Expresión Regular
<b>Identificador</b>	[a-zA-Z][a-zA-Z0-9]*
<b>Palabra Reservada</b>	use dropDatabase createCollection dropCollection insertOne updateOne deleteOne find findOne BuscarTodo BuscarUnico CrearColeccion EliminarColeccion
<b>Paréntesis abierto</b>	(
<b>Paréntesis cerrado</b>	)
<b>Operador asignación</b>	=

<b>Punto y coma</b>	;
<b>Cadena de caracteres</b>	"[^"]*"

## Diagrama de Árbol



## Autómata Finito Determinista

El AFD desempeña un papel fundamental en el análisis léxico del texto de entrada, ya que permite reconocer y clasificar los diferentes tokens que conforman el lenguaje específico utilizado por el traductor.

El AFD implementado en el proyecto del Traductor HTML consta de un conjunto finito de estados, un alfabeto de entrada, una función de transición y un conjunto de estados de aceptación. A continuación, se describen los componentes principales del AFD:

### Conjunto de Estados (N)

- El conjunto de estados del AFD representa los diferentes estados en los que puede encontrarse el autómata durante el proceso de análisis léxico.



- Cada estado representa una etapa específica del proceso de reconocimiento de tokens, desde el estado inicial hasta los estados de aceptación.

$$N = \{q0, q1, q2, q3, q4, q5, q6, q7\}$$

### Alfabeto de Entrada ( $\Sigma$ )

- El alfabeto de entrada del AFD incluye todos los caracteres válidos que pueden formar parte del texto de entrada.
- Este alfabeto define los símbolos que el AFD puede reconocer y procesar durante el análisis léxico.

$$\Sigma = \{I, S, N, C, P, R\}$$

$$I = \left\{ \begin{array}{l} CrearBD, EliminarBD, CrearColeccion, CrearColeccion, \\ EliminarColeccion, InsertarUnico, ActualizarUnico, \\ EliminarUnico, BuscarTodo, BuscarUnico \end{array} \right\}$$

$$S = \{=\} \quad N = \{nueva\} \quad C = \{Cadena\}$$

$$P = \{(,)\} \quad R = \{;\}$$

### Estado Inicial ( $S_0$ ):

El estado inicial del AFD representa el punto de partida del autómata en el proceso de análisis léxico.

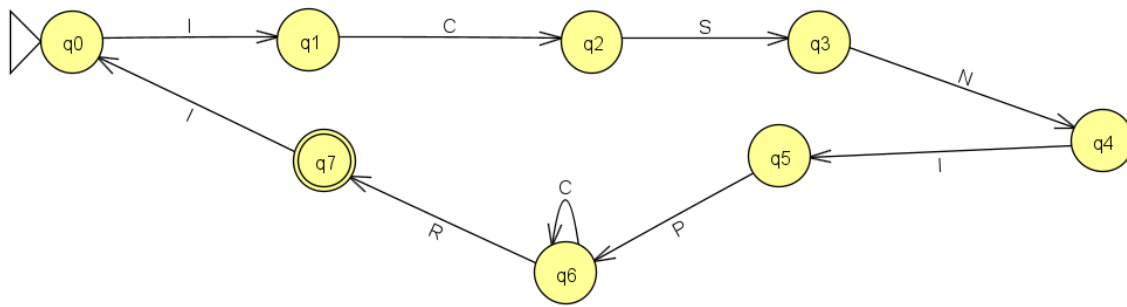
Desde el estado inicial, el AFD comienza a procesar el texto de entrada y a reconocer los tokens presentes en él.

$$Inicio = Q_0$$

### Función de Transición

- La función de transición del AFD define cómo el autómata se mueve de un estado a otro en respuesta a la entrada proporcionada.
- Para cada estado y símbolo de entrada, la función de transición determina el próximo estado al que el AFD debe transicionar.

$q_0 \rightarrow lq_1$   
 $q_1 \rightarrow Cq_2$   
 $q_2 \rightarrow Sq_3$   
 $q_3 \rightarrow Nq_4$   
 $q_4 \rightarrow lq_5$   
 $q_5 \rightarrow Pq_6$   
 $q_5 \rightarrow Cq_6 \mid Rq_7$   
 $q_7 \rightarrow lq_0 \mid \epsilon$



## Gramática Libre de Contexto

1. Instrucción: CrearBD | EliminarBD | CrearColeccion | EliminarColeccion | InsertarUnico | ActualizarUnico | EliminarUnico | BuscarTodo | BuscarUnico
2. CrearBD: "CrearBD" Identificador "=" "nueva" "CrearBD" "(" ")" ";"
3. EliminarBD: "EliminarBD" Identificador "=" "nueva" "EliminarBD" "(" ")" ";"
4. CrearColeccion: "CrearColeccion" Identificador "=" "nueva" "CrearColeccion" "(" Cadena ")" ";"
5. EliminarColeccion: "EliminarColeccion" Identificador "=" "nueva" "EliminarColeccion" "(" Cadena ")" ";"
6. InsertarUnico: "InsertarUnico" Identificador "=" "nueva" "InsertarUnico" "(" Cadena "," Cadena ")" ";"

7. ActualizarUnico: "ActualizarUnico" Identificador "=" "nueva" "ActualizarUnico" "(" Cadena "," "{" "\$set" ":" "{" Cadena ":" Cadena "}" "}" ")" ";"
8. EliminarUnico: "EliminarUnico" Identificador "=" "nueva" "EliminarUnico" "(" Cadena "," "{" Cadena ":" Cadena "}" "}" ")" ";"
9. BuscarTodo: "BuscarTodo" Identificador "=" "nueva" "BuscarTodo" "(" Cadena ")" ";"
10. BuscarUnico: "BuscarUnico" Identificador "=" "nueva" "BuscarUnico" "(" Cadena ")" ";"

## Traducción a MongoDB

La traducción a sentencias MongoDB se lleva a cabo mediante el analizador léxico, que toma los tokens identificados en el análisis léxico y de texto y genera las sentencias correspondientes.

Los diferentes elementos como CrearDB, EliminarDB, CrearColeccion, etc., se traducen a sentencias de MongoDB apropiadas.

Se aplican estilos CSS correspondientes, como color de texto, tamaño de fuente, alineación, etc.

```

1  if tipo_funcion == 'CrearBD':
2      nombre_bd = variable_asignacion.strip().split('=', 1)[-1].strip("\"'")
3      salida_final = f"{palabras_reservadas[tipo_funcion]} ({nombre_bd});"
4      sentencias_generadas.append(salida_final)
5  elif tipo_funcion == 'EliminarBD':
6      salida_final = f"{palabras_reservadas[tipo_funcion]}();"
7      sentencias_generadas.append(salida_final)
8  elif tipo_funcion == 'EliminarColeccion':
9      inicio_comillas = parte_funcion.find('(') + 1
10     fin_comillas = parte_funcion.find(')', inicio_comillas)
11     if inicio_comillas != -1 and fin_comillas != -1:
12         nombre_coleccion = parte_funcion[inicio_comillas:fin_comillas].strip("\"'")
13         salida_final = f"db.{nombre_coleccion}.drop();"
14         sentencias_generadas.append(salida_final)

```

```

1 elif tipo_funcion == 'CrearColeccion':
2     inicio_comillas = parte_funcion.find('(') + 1
3     fin_comillas = parte_funcion.find(')', inicio_comillas)
4     if inicio_comillas != -1 and fin_comillas != -1:
5         nombre_coleccion = parte_funcion[inicio_comillas:fin_comillas].strip('\n')
6         salida_final = f"db.createCollection('{nombre_coleccion}');"
7         sentencias_generadas.append(salida_final)

```

```

1 elif tipo_funcion == 'BuscarTodo':
2     inicio_comillas = parte_funcion.find('(') + 1
3     fin_comillas = parte_funcion.find(')', inicio_comillas)
4     if inicio_comillas != -1 and fin_comillas != -1:
5         nombre_coleccion = parte_funcion[inicio_comillas:fin_comillas].strip('\n')
6         salida_final = f"db.{nombre_coleccion}.find();"
7         sentencias_generadas.append(salida_final)

```

```

1 elif tipo_funcion == 'BuscarUnico':
2     inicio_comillas = parte_funcion.find('(') + 1
3     fin_comillas = parte_funcion.find(')', inicio_comillas)
4     if inicio_comillas != -1 and fin_comillas != -1:
5         nombre_coleccion = parte_funcion[inicio_comillas:fin_comillas].strip('\n')
6         salida_final = f"db.{nombre_coleccion}.findOne();"
7         sentencias_generadas.append(salida_final)

```

## Generación de Resultados

El resultado generado se muestra en el área de texto final para que el usuario pueda ver y copiar las sentencias generadas.

Se generan archivos HTML adicionales para registrar los lexemas y errores encontrados durante el análisis léxico.

```

1 def imprimirLexemas(lexemas, errores):
2     with open("lexemas.html", "w", encoding='utf-8') as f:
3         f.write("<html>\n<head>\n<title>Listado de Tokens y Lexemas</title>\n</head>\n<body>\n")
4         f.write("<h1>Listado de Tokens y Lexemas</h1>\n")
5         f.write("<table border='1'>\n")
6         f.write("<tr><th>Correlativo</th><th>Token</th><th>No. de Token</th><th>Lexema</th></tr>\n")
7
8         for idx, lexema in enumerate(lexemas, start=1):
9             f.write(f"<tr><td>{idx}</td><td>{lexema.tipo}</td><td>{idx}</td><td>{lexema.lexema}</td></tr>\n")
10
11         f.write("</table>\n")
12         f.write("</body>\n</html>")
13
14     webbrowser.open('file://' + os.path.realpath("lexemas.html"))

```

```

1 def generar_tabla_errores(errores):
2     print("Generando tabla de errores...")
3     with open("errores.html", "w", encoding='utf-8') as f:
4         f.write("<html>\n<head>\n<title>Errores</title>\n</head>\n<body>\n")
5         f.write("<h1>Errores Lexicos y Sintacticos</h1>\n")
6         f.write("<table border='1'>\n")
7         f.write("<tr><th>Tipo de Error</th><th>Linea</th><th>Columna</th><th>Token</th></tr>\n")
8
9         for error in errores:
10             token_esperado = error.token_esperado if error.token_esperado is not None else ""
11             f.write(f"<tr><td>{error.tipo}</td><td>{error.fila}</td><td>{error.columna}</td><td>{token_esperado}</td></tr>\n")
12
13         f.write("</table>\n")
14         f.write("</body>\n</html>")
15     webbrowser.open('file://' + os.path.realpath("errores.html"))
16
17

```

## Estructura del proyecto

El proyecto sigue una estructura típica de Python:

- TraductorMongoDB
  - main.py: Script principal para ejecutar la aplicación.
    - lexema.py: contiene las clases Lexema y Error
    - analizadorLexico.py: se analizan los caracteres válidos del lenguaje
    - analizadorSintactico.py: se analiza el texto ingresado sintácticamente.

## Manejo de Errores

---

- Detección de Errores en el Código Fuente:

Durante el desarrollo del proyecto, se emplean buenas prácticas de programación para evitar la introducción de errores en el código.

Se realizan pruebas exhaustivas y revisiones de código para identificar y corregir posibles errores antes de la implementación final del software.

## Áreas de Mejoras Futuras

---

Se considerará en el futuro mejorar el código para que pueda capturar y generar las sentencias para InsertarUnico, ActualizarUnico, EliminarUnico y los comentarios. Así como añadir los errores sintácticos en la tabla de errores que se genera.