DOCUMENTACIÓN MINIMAL REACT CMS

A continuación se presenta la documentación del proyecto de Minimal React CMS, se presenta la estructura a través de un diagrama de clases, y explicación de cada clase que compone el proyecto dividido en backend y frontend.

Tecnologías utilizadas

Se utilizó las siguientes tecnologías para el desarrollo de la aplicación.

- JDK 22 tanto para el backend y el frontend.
- Jflex para la creación del analizador léxico, jar utilizado: jflex-full-1.9.1.jar.
- Cup para la creación del analizador sintáctico por medio de gramáticas, jar utilizado: java-cup.11b.jar.
- Antlr4 para analizador tanto léxico como sintáctico versión 4.13.2.
- WebSockets para el servidor backend.
- JavaFx para el frontend.
- Intellij IDEA como IDE para la creación de la aplicación backend y frontend.
- PlantUML para crear el diagrama de clases.

Diagrama de clases

El siguiente diagrama de clases contiene se divide en dos partes, una muestra cómo se estructura el servidor backend y la otra muestra cómo se estructura el frontend.

Diagrama de clases del servidor (backend)

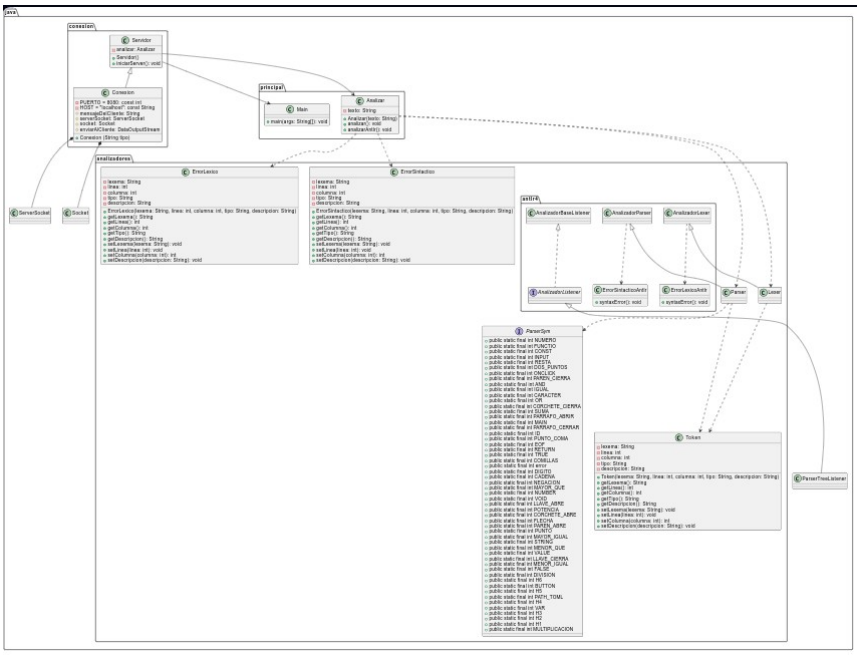
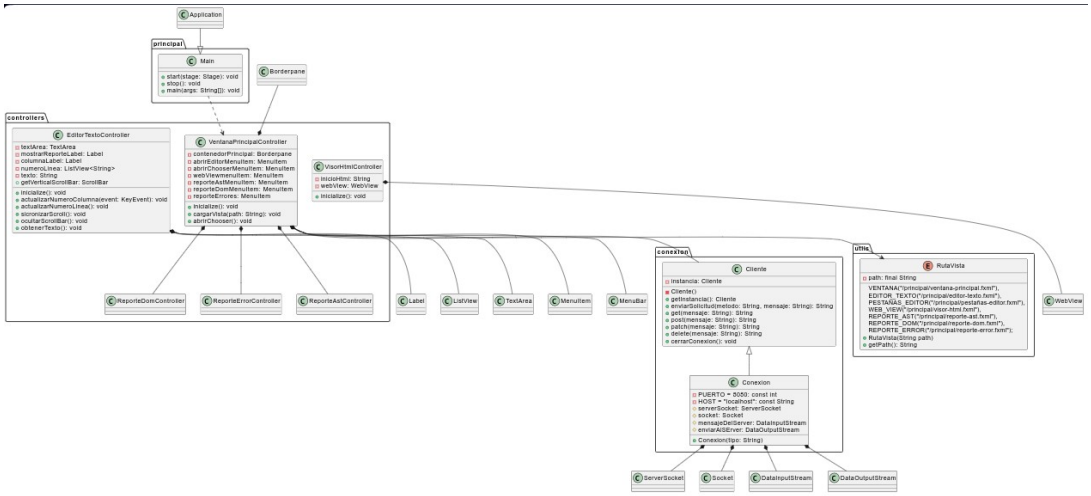


Diagrama de clases de la aplicación de usuario (frontend)



Descripción de clases de cada diagrama

A continuación se describe el funcionamiento de cada clase que compone el backend y el fronted.

Backend

El backend es un servidor el cuál se comunica por medio de WebSockets, se compone de lo siguiente:

Paquete antlr4

Este paquete contiene la carpeta que contiene la carpeta analizadores y este la carpeta antlr4, dentro de esta carpeta está el archivo Analizador.g4, esta estructura de carpetas coincide con la misma de la carpeta java, esto es necesario para que se pueda utilizar el analizador léxico y sintáctico de antlr4.

Paquete cup

Este paquete contiene el archivo parser.cup que es el encargado de generar las gramáticas que deben cumplir los archivos de texto.

Paquete java

Este paquete contiene las clases java y paquetes que permitirán el funcionamiento del servidor WebSocket.

Paquete analizadores

Este paquete contiene otro paquete donde se encuentra el paquete antlr4 que contiene las clases ErrorLexicoAntlr y ErrorSintacticoAntlr.

Clase AnalizadorBaseListener

Esta clase se encarga de contener métodos los cuales son generados por antlr4 para que puedan sobrescribirse según las necesidades del desarrollador.

ClaseAnalizadorLexer

Esta clase es generada por antlr4 y contiene métodos para manejar los tokens del texto ingresado.

Clase AnalizadorListener

Esta es una interface generada por antlr4 para que los tokens se almacenen y puedan ser usados por el parser.

Clase AnalizadorParser

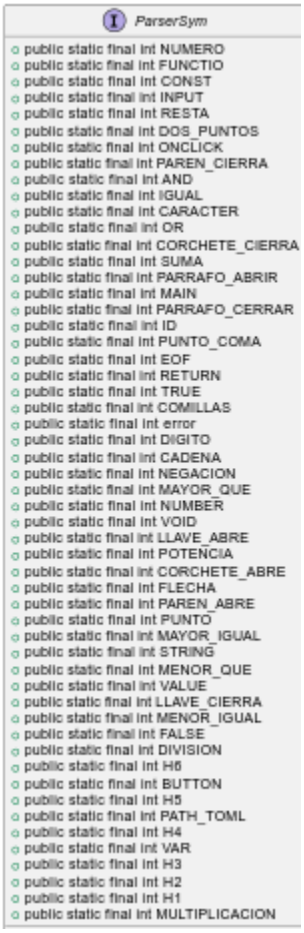
Esta clase es generada por antlr4 y se encarga de trabajar con las producciones definidas en el archivo Analizadores.g4.

Clase Parser

Esta clase es generada por cup y se encarga de manejar las producciones que el desarrollador ingresó en el archivo parser.cup.

Clase ParserSym

Esta clase la genera cup y contiene los tokens que serán utilizados por el parser.

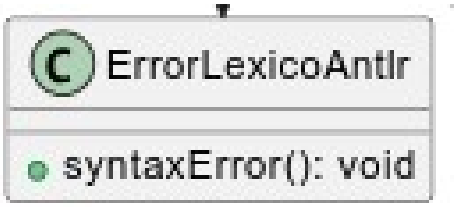


Clase Lexer

Esta clase es generada por jflex y se encarga de recibir el texto ingresado por el usuario y crear y detectar los tokens necesarios para el parser.

Clase ErrorLexicoAntlr

Esta clase se encarga de añadir todos los errores léxicos que puedan existir dentro del análisis.

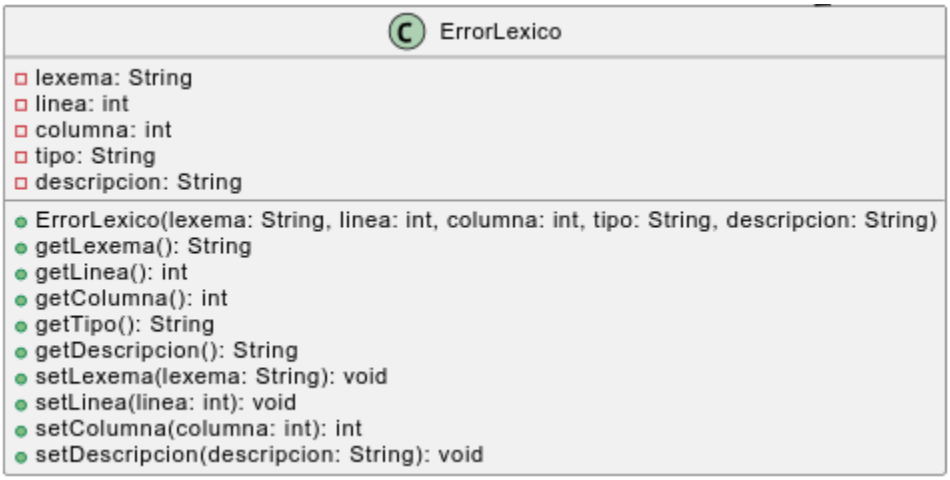


Clase ErrorSintacticoAntlr

Esta clase se encarga de añadir todos los errores sintácticos que puedan existir dentro del análisis.

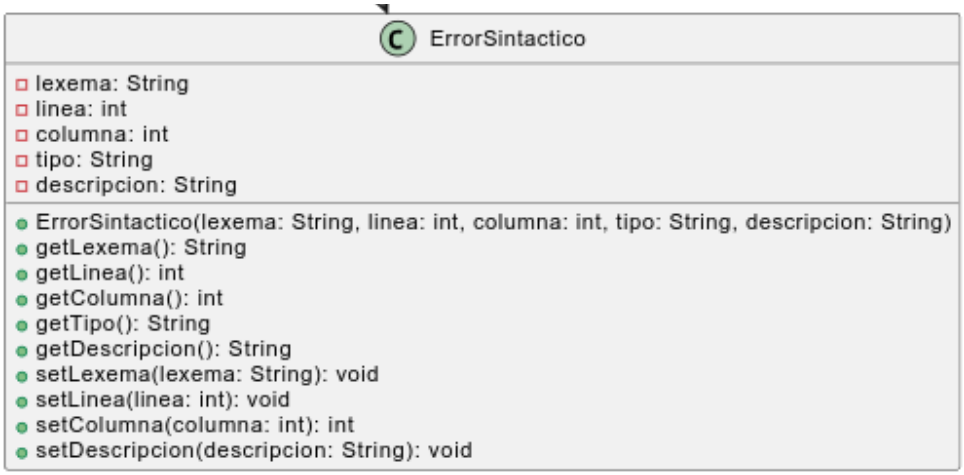
Clase ErrorLexico

Esta clase sirve únicamente para tener el molde de un error léxico.



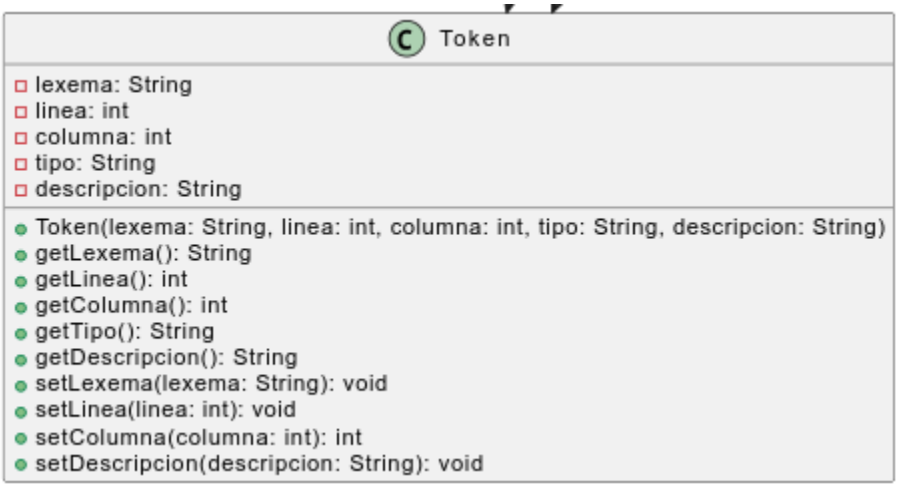
Clase ErrorSintactico

Esta clase sirve únicamente para tener el molde de un error sintáctico.



Clase Token

Esta clase sirve para ser el modelo de un token que contiene número de línea, columna, tipo y lexema.

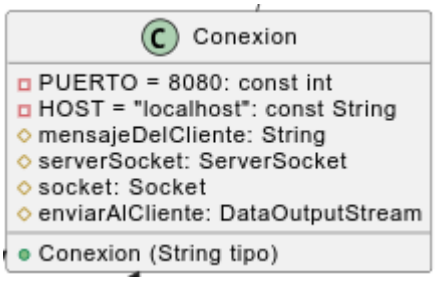


Paquete conexión

Este paquete contiene información sobre las clases que permiten que sea un servidor socket.

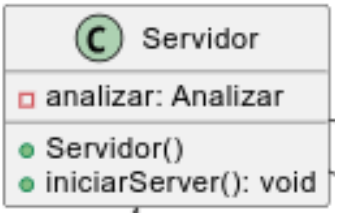
Clase Conexion

Esta clase se encarga de ser quién defina los parámetros de conexión.



Clase Servidor

Esta clase se encarga de permitir la conexión que estará activa para que se conecten los clientes.

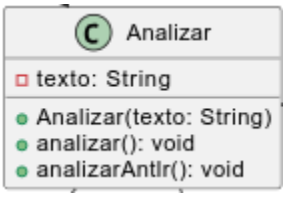


Paquete principal

Este paquete contiene las clases que permiten la ejecución del servidor además del análisis.

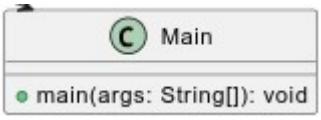
Clase Analizar

Esta clase se encarga de hacer uso del lexer y parser que funcionaran para analizar el texto enviado por el frontend.



Clase Main

Esta clase se encarga de iniciar el servidor haciendo llamados en el método main a las clases necesarias.



Frontend

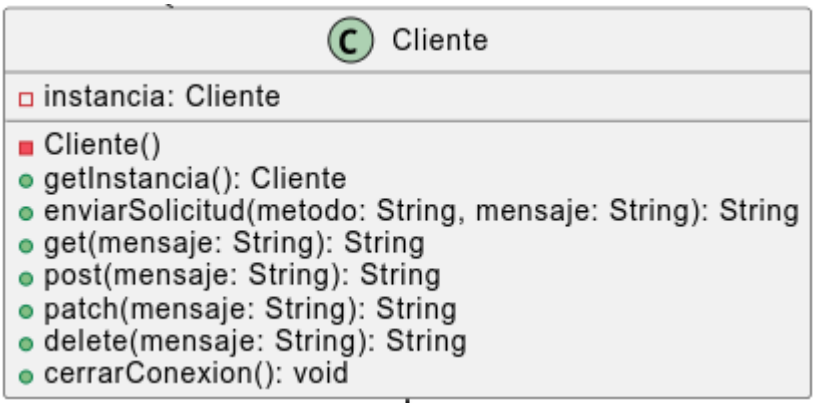
El frontend es una aplicación con JavaFx el cual se compone de lo siguiente.

Paquete conexión

Este paquete se encarga de la conexión entre cliente-servidor, contiene las siguientes clases.

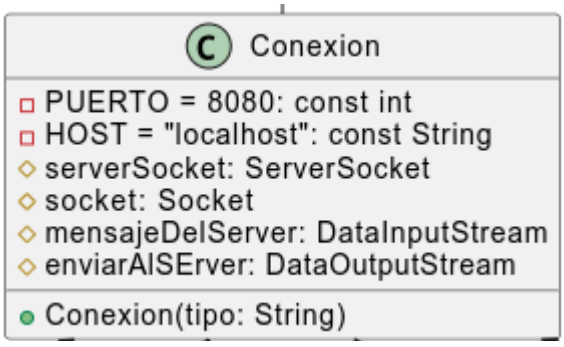
Clase Cliente

La clase Cliente contiene métodos la cuál sirve para conectarse al servidor.



Clase Conexion

La clase es la base en la cuál contiene la información necesaria para que conecte con el servidor.

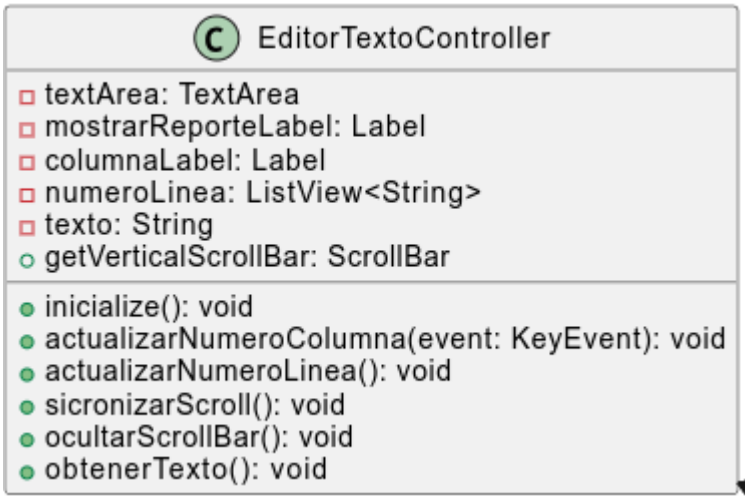


Paquete controllers

Este paquete contiene los controladores de los archivos FXML que conformarán las vistas.

EditorTextoController

Esta clase contiene la lógica que hace que se muestre tanto el editor de texto como la vista que muestra el número de filas.



ReporteAstController

Este controlador se encarga de mostrar los reportes AST.

ReporteDomController

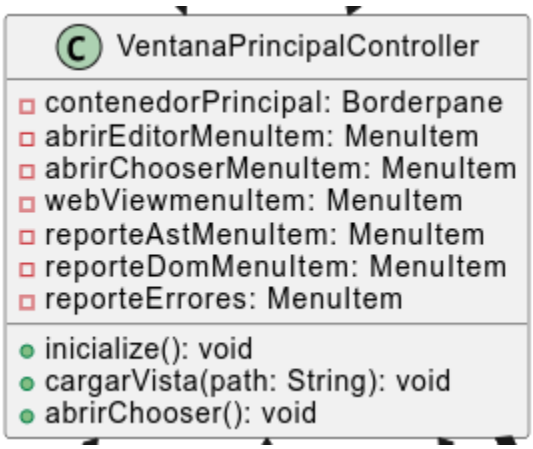
Este controlador se encarga de mostrar los reportes DOM

ReporteErrorController

Esta parte se encarga de mostrar los reportes de analisis léxicos o sintácticos que puedan existir dentro del programa.

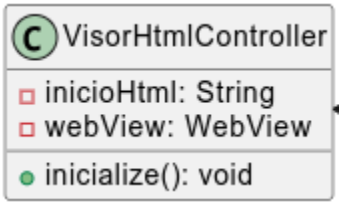
VentanaPrincipalController

Esta clase se encarga de mostrar todos los componentes que conforman la aplicación y cambiar las vistas, según lo requiera el usuario.



VisorHtmlController

Esta vista se encarga de mostrar texto HTML que sea generado por el usuario.

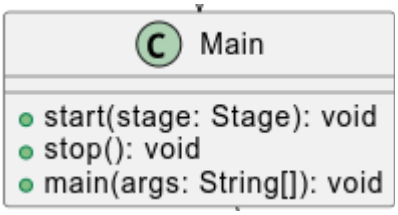


Paquete principal

Este paquete contiene únicamente la clase Main

Clase Main

Esta extiende de Aplication y se encarga de iniciar la ejecución de la aplicación

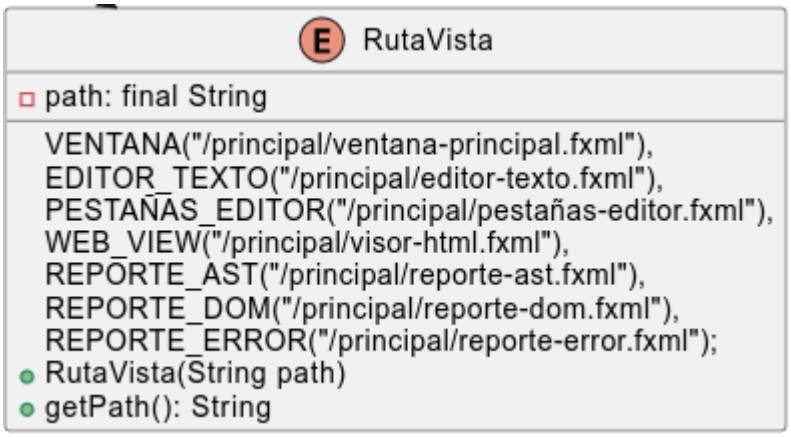


Paquete utils

Este paquete contiene una clase la cuál se utiliza para almacenar las rutas de los archivos FXML.

Clase RutaVista

Esta clase es una clase enum la cual almacena las rutas de los archivos FXML para que sea más fácil manejarlos.



Gramáticas

Las gramáticas utilizadas en el parser.cup son las siguientes:

Terminales:

Los terminales son los tokens donde las gramáticas terminarán, es decir, al ingresar algún texto estos se comparan con los terminales para definir que el texto ingresado es correcto, las terminales definidas para este analizador son las siguientes:

get, post, patch, delete, success, not_found, internal_server_error, sitio, pagina, SITIO, PAGINA, crear, agregar, eliminar, modificar, nombre, path, true, false, void, main, const, var, string, number, boolean, char, function, return, console, log, if, else, for, h1, h2, h3, h4, h5, h6, , , input, value, button, onClick, [,], {, }, (,), “, +, -, *, /, ^, =, <, >, <=, =>, >=, ||, &&. !, ., ,, ;, :, digito, id, numero, cadena, carácter, path_toml, comentario_toml, comentario_linea, comentario_bloque.

No terminales:

Los no terminales son aquellos que nos llevan a otra producción, contienen terminales y no terminales dentro de una producción, los no terminales definidos son:

inicial, request, response, toml, mini_react, metodos, objetivo_shttp, scl_intstruccion, accion, objetivo_scl, parametros, response, codigo, body_html, contenido, header, parrafos, inputs, buttons, texto, toml, etiqueta_toml, atributo_toml, mini_react, cuerpo_react, tipo_variable, asignacion_variable, asignacion_numero, funciones, parametros_fun, bloque_codigo, if, for, condiciones, return_react, calculadora.

Producciones

Es la producción INICIAL en la cual se encarga de definir qué producción continuará según vaya encontrando los tokens.

INICIAL → REQUEST

| RESPONSE

| TOML

| MINI_REACT

La producción REQUEST se encarga de recibir peticiones como get, post, patch, delete, además recibe un body html.

REQUEST : METODOS OBJETIVO_SHTTP SCL_INSTRUCCION BODY_HTML

La producción METODOS es para definir parte del cuerpo de un request.

METODOS → get

| post

| patch

| delete

La producción OBJETIVO_SHTTP define que tipo de request será.

OBJETIVO_SHTTP → sitio_shttp

| pagina_shttp

La producción SCL_INSTRUCCION se encarga de contener más producciones en las cuales derivan en otras producciones para componer un request.

SCL_INSTRUCCION → ACCION OBJETIVO_SCL PARAMETROS

La producción ACCION contiene terminales que son acciones dentro de una instrucción scl.

ACCION → crear

| agregar

| eliminar

| modificar

La producción OBJETIVO_SCL contiene terminales que conformarán la instrucción scl.

OBJETIVO_SCL → sitio_scl

| pagina_scl

La producción PARAMETROS contiene un terminal y es recursiva, ya que puede permitir la entrada de un parámetro o más parámetros.

PARAMETROS → id

| id PARAMETROS

La producción RESPONSE representa una respuesta HTTP, compuesta por dos producciones.

RESPONSE : CODIGO BODY_HTML

La producción CODIGO define códigos de respuesta HTTP posibles.

CODIGO → success

| not_found

| internal_server_error

La producción BODY_HTML representa el cuerpo de una respuesta en formato HTML.

BODY_HTML → menor_que main mayor_que CONTENIDO menor_que division main mayor_que

| menor_que main mayor_que CONTENIDO (CONTENIDO)* menor_que division main mayor_que

La producción CONTENIDO es un conjunto de producciones que pueden ser headers, parrafos, inputs, buttons.

CONTENIDO → HEADER

| PARRAFOS

| INPUTS

| BUTTONS

La producción HEADER representa 6 diferentes tipos de etiquetas dentro de un HTML.

HEADER → menor_que h1 mayor_que TEXTO menor_que diagonal h1 mayor_que

| menor_que h2 mayor_que TEXTO menor_que diagonal h2 mayor_que

| menor_que h3 mayor_que TEXTO menor_que diagonal h3 mayor_que

| menor_que h4 mayor_que TEXTO menor_que diagonal h4 mayor_que

| menor_que h5 mayor_que TEXTO menor_que diagonal h5 mayor_que

| menor_que h6 mayor_que TEXTO menor_que diagonal h6 mayor_que

La producción PARRAFOS representa una etiqueta de parrafos en HTML.

PARRAFOS → parrafo_abrir TEXTO parrafo_cerrar

La producción INPUTS representa la entrada en una etiqueta HTML.

INPUTS → menor_que input value igual llave_abre id llave_cierra diagonal mayor_que

La producción BUTTONS representa un botón dentro de HTML.

BUTTONS → menor_que button onclick igual llave_abre id llave_cierra mayor_que TEXTO menor_que diagonal button mayor_que

La producción TEXTO es el contenido que puede tener un botón, párrafo o texto dentro de HTML.

TEXTO → id

| id TEXTO

| TEXTO llave_abre id llave_cierra

| TEXTO llave_abre id llave_cierra TEXTO

La producción TOML representa el lenguaje toml.

TOML → ETIQUETA_TOML ATRIBUTO_TOML (TOML)*

La producción ETIQUETA_TOML es una producción que será utilizada en la producción TOML.

ETIQUETA_TOML → corchete_abre id corchete_cierra

| corchete_abre id (punto id)+ corchete_cierra

La producción ATRIBUTO_TOML es una producción que representa una asignación de variables.

ATRIBUTO_TOML → nombre_toml igual cadena

| path igual path_toml

| nombre_toml igual cadena path igual path_toml

La producción MINI_REACT representa la producción de un lenguaje combinado entre html y type script.

MINI_REACT → const ID igual paren_abre paren_cierra flecha llave_abre CUERPO_REACT RETURN_REACT llave_cierra

La producción CUERPO_REACT es parte del lenguaje mini react.

CUERPO_REACT → var ID dos_puntos TIPO_VARIABLE igual ASIGNACION_VARIABLE punto_coma

| (var ID dos_puntos TIPO_VARIABLE igual ASIGNACION_VARIABLE punto_coma)+

| (var ID dos_puntos TIPO_VARIABLE igual ASIGNACION_VARIABLE punto_coma)+ FUNCIONES+

| FUNCIONES+

La producción TIPO_VARIABLE representa las diferentes tipos de variables que se pueden asignar en el lenguaje.

TIPO_VARIABLE → number

| string

| char

| boolean

La producción ASIGNACION_VARIABLE representa los valores que se pueden asignar a un tipo de variable.

ASIGNACION_VARIABLE → ASIGNACION_NUMERO

| cadena

| caracter

| (true | false)

La producción ASIGNACION_NUMERO representa las operaciones que se puede asignar a un tipo de variable de tipo number.

ASIGNACION_NUMERO → numero

| CALCULADORA

La producción FUNCIONES es una regla que debe cumplir una función para que pueda existir.

FUNCIONES → function ID paren_abre (PARAMETROS_FUN)? paren_cierra dos_puntos void llave_abre BLOQUE_CODIGO llave_cierra

| function ID paren_abre (PARAMETROS_FUN)? paren_cierra dos_puntos TIPO_VARIABLE llave_abre BLOQUE_CODIGO return CALCULADORA llave_cierra

La producción PARAMETROS_FUN se encarga de mostrar o no parámetros dentro de una función ya que estos pueden tener parámetros.

PARAMETROS_FUN → ID dos_puntos (TIPO_VARIABLE | void) (coma ID dos_puntos (TIPO_VARIABLE | void))+

La producción BLOQUE_CODIGO se encarga de tener el contenido dentro de un bloque de código.

BLOQUE_CODIGO → IF

| FOR

La producción IF representa la estructura de una sentencia if en cualquier lenguaje.

IF → if paren_abre numero paren_cierra llave_abre llave_cierra (else if paren_abre numero paren_cierra llave_abre llave_cierra)*

| if paren_abre numero paren_cierra llave_abre llave_cierra else llave_abre llave_cierra

La producción FOR es la estructura de un ciclo for en cualquier lenguaje.

FOR → for paren_abre ASIGNACION_NUMERO punto_coma CONDICIONES punto_coma ('I++' | 'I--') paren_abre llave_cierra BLOQUE_CODIGO llave_cierra

La producción CONDICIONES representa la condición dentro de una sentencia if que esta en la producción anterior.

CONDICIONES → CALCULADORA mayor_que CALCULADORA

| CALCULADORA menor_que CALCULADORA

| CALCULADORA mayor_igual CALCULADORA

| CALCULADORA menor_igual CALCULADORA

La producción RETURN_REACT representa la parte en la que se retornará html para que sea mostrado posteriormente.

RETURN_REACT → return paren_abre BODY_HTML paren_cierra punto_coma

La producción CALCULADORA se encarga de las operaciones aritméticas, esta se llama a si misma en la mayoría de sus producciones ya que puede haber varias operaciones aritméticas en una misma expresión ingresada por el usuario, también se considera que el usuario ingrese únicamente un número

se muestra a continuación:

CALCULADORA → resta CALCULADORA

| paren_abre CALCULADORA paren_cierra

| CALCULADORA potencia CALCULADORA

| CALCULADORA (multiplicacion | division) CALCULADORA

| CALCULADORA (suma | resta) CALCULADORA

| numero