Manual Técnico “Mini-C”

A continuación se presenta la documentación técnica del proyecto Mini-C, se mostrará el diagrama de clases, además de describir que hace cada método que en ellas contiene. Esta es una aplicación completamente web.

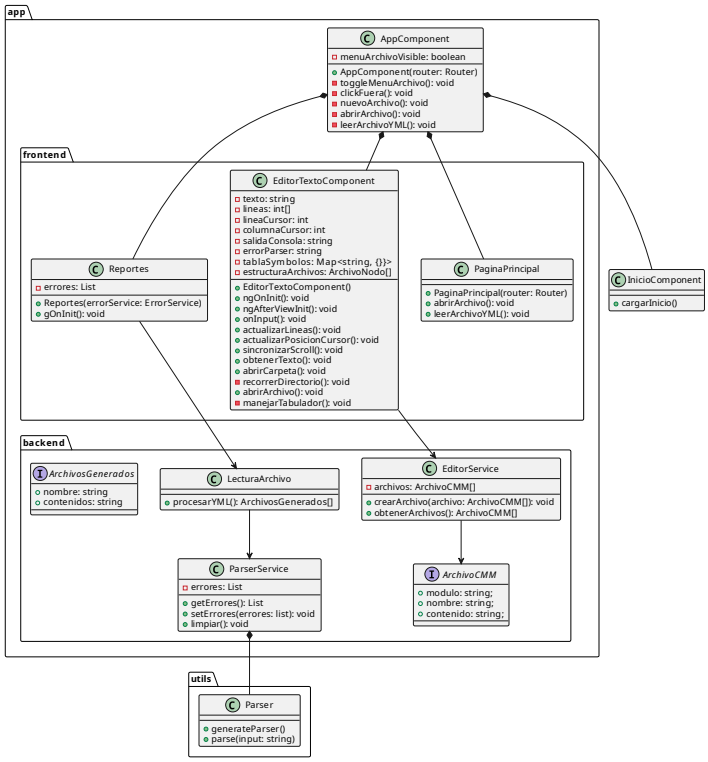
Tecnologías utilizadas

Para el desarrollo de la aplicación se utilizó lo siguiente:

- Angular CLI 19.2.7 como lenguaje de programación.
- Peggy ^4.2.0 para el analizador léxico, sintáctico y semántico.
- PlantUml para generar el diagrama de clases.

Diagrama de clases

A continuación se muestra el diagrama de clases que compone el programa, este diagrama contiene las relaciones entre las clases que permite su funcionamiento.

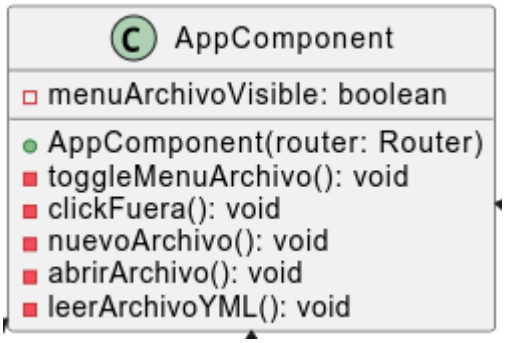


Paquete app

Este paquete contiene toda la estructura del proyecto, acá están las subcarpetas y archivos ts, html y css que permiten las vistas y funcionamiento de la aplicación, las carpetas y archivos se describen a continuación.

Clase App

Esta clase es la que se encarga de iniciar toda la aplicación, acá se comienza el flujo.

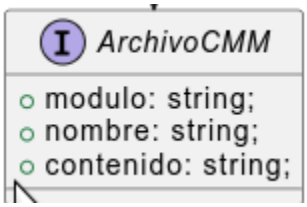


Paquete backend

Este paquete contiene clases que permiten funcionalidades para la creación de la vista de los módulos y archivos cmm.

Clase ArchivoCMM

Esta es una interface que permite la creación de modulos nombre y el contenido.

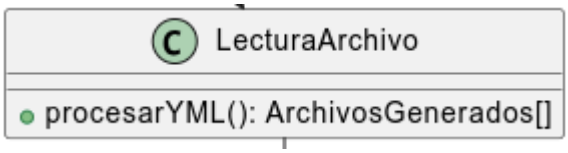


Clase EditorService

Este permite pasar el contenido de un archivo cmm al editor de texto.

Clase LecturaArchivo

Esta clase se encarga de procesar un archivo “config.yml”, que contiene la configuración de un proyecto, contiene módulos, archivos cmm y la clase main.



Paquete frontend

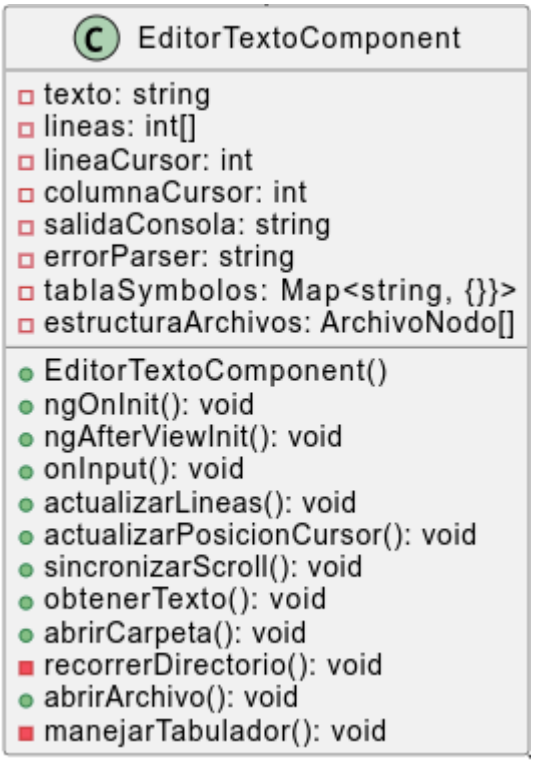
Este paquete contiene los archivos html y los ts que funcionan para la interfaz de usuario. Cada vista esta dentro de una capeta para mantener su modularización y archivos que lo componen en una carpeta organizada.

Paquete editor

En este paquete se encuentran los tres archivos que componen una vista, el archivo ts, html y css, pero a continuación se describe únicamente el archivo ts.

Clase editor-texto.component

Esta clase se encarga de gestionar las acciones del editor de texto, a continuación se describen sus métodos.



Método ngOnInit
Se encarga de mostrar en la vista de usuario el número de líneas haciendo una llamada al método que contiene la lógica para capturar eso.

Método ngAfterViewInit
Se encarga de actualizar la posición del puntero que indicará la columna haciendo una llamada al método que contiene la lógica para capturar eso.

Método onInput
Este método se encarga de hacer el llamado a los métodos para actualizar la posición del puntero indicando el número de línea y columna.

Método actualizarLinea
Este método contiene la lógica para obtener el número de línea, se basa en contar los saltos de línea y en base a ello usa un contador para definier el número de linea.

Método actualizarPosicionCursor

Este se encarga de obtener y mostrar el número de columna en el que se encuentra el puntero, se basa convirtiendo los caracteres de una línea en un arreglo y así obtener el número de columna.

Método sincronizarScroll

Este método se encarga de sincronizar el scroll del área que muestra el número de línea y el área de texto.

Método obtenerTexto

Este método obtiene el texto del área de texto y posteriormente lo parsea y según el resultado obtenido lo muestra en la consola destinada en la interfaz de usuario.

Método abrirCarpeta

Este método se encarga de abrir un pop up para buscar el archivo “config.yml” y posteriormente lo parsea para crear los módulos y archivos cmm.

Paquete paginaPrincipal

En este paquete están los archivos para la vista principal, pero solo se describirá el archivo ts.

Clase inicio.component.ts

En esta clase únicamente permite abrir un archivo nuevo o crear un archivo para proyecto nuevo, a continuación se describen los métodos.

Método abrirArchivo

Este método se encarga de leer el archivo yml para crear el proyecto.

Método leerArchivoYML

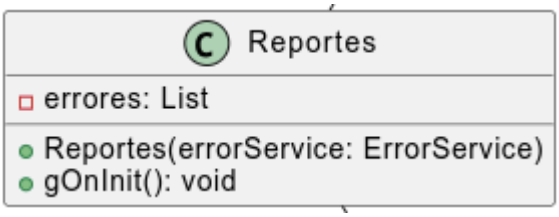
Este método lo lee y parsea para crear el proyecto de Mini-C,

Paquete reportes

Este paquete se encarga de mostrar los reportes de los parseos.

Clase reportes.component.ts

Esta clase genera los reportes del parseo o los parseos realizados en la ejecución del programa.



Paquete utils

En este paquete se encuentran los archivos generados por el parser peggy, este genera los archivos parser,js y parser.d.ts. Lo que contiene no es legible pero las gramáticas se explican en el siguiente apartado.

Gramáticas

A continuación se describen las gramáticas que se utilizaron para poder analizar tanto el lenguaje yml como mini-c.

Terminales

Los terminales son aquellos que indican un valor en concreto, es decir, que indican que la producción ha llegado a un valor final, los terminales usados son:

No terminales

Estos indican que la producción no termina, es decir, indica que es otra producción, los terminales no usados son:

Producciones

Las producciones se utilizan para parsear un texto según las reglas definidas por las gramáticas, a continuación se presentan las gramaticas para yaml y mini-c.

Gramáticas iniciales

Estas producciones son las iniciales y definen que lenguaje van a parsear.

inicio → lenguajes

lenguajes → estructuraYAML
| miniC:miniC

Gramáticas para YAML

Las gramáticas para yaml son las siguientes:

ESTRUCTURA_YAML → NOMBRE_PROYECTO MAIN MODULOS*

NOMBRE_PROYECTO → COMENTARIO* "NOMBRE_PROYECTO" ":" "VARIABLE " COMENTARIO*

MAIN → COMENTARIO* "MAIN" ":" VARIABLE ".cmm" COMENTARIO*

MODULOS → COMENTARIO* "modulo" NUMERO ":" "\n" LISTA* COMENTARIO*

LISTA → COMENTARIO* "archivo" NUMERO ":" VARIABLE ".cmm" "\n"* COMENTARIO*

COMENTARIO → "#" [^\n]* ("\n" / !.)

Gramáticas para Mini-C

Las producciones para detectar el lenguaje mini-c es la siguiente

MINI_C → METODO_MAIN CUERPO_MINI_C*

METODO_MAIN → "void" "main" "(" ")" "{" SENTENCIA* "}"

SENTENCIA→ DECLARACION_VARIABLES
| ASIGNACION_VARIABLE
| CONDICIONAL
| IMPRESION_CONSOLA
| FUNCIONES
| :STRUCTS

```
DECLARACION_VARIABLES → tipoVariable _ variable:variable _ "=" _ valor:expresion _ ";"

ASIGNACION_VARIABLES → VARIABLE "=" EXPRESION ";"
                     | VARIABLE "=" FUNCION

FUNCION → TIPO_VARIABLE VARIABLE "(" PARAMETROS* ")" "{"
CONTENIDO_FUNCION* "return" VARIABLE ";" "}"
        | "void" variable "(" PARAMETROS:PARAMETROS* ")" "{" CONTENIDO_FUNCION*
"}"

TIPO_VARIABLE → "int"
              | "float"
              | "string"
              | "char"
              | "bool"

EXPRESION → SUMA_RESTA

SUMA_RESTA → MULTIPLICACION_DIVISION ("+" | "-") SUMA_RESTA
           | MULTIPLICACION_DIVISION

MULTIPLICACION_DIVISION → POTENCIA ("*" | "/") MULTIPLICACION_DIVISION

MULTIPLICACION_IMPLICITA → POTENCIA

POTENCIA → UNARIO "^" POTENCIA
         | UNARIO

UNARIO → "-" UNARIO
       | PRIMARIO

PRIMARIO → "(" SUMA_RESTA ")"
         | NUMERO
         | VARIALBE
         | CADENA
         | CARACTER
         | BOOLEAN

CUERPO_MINIC → STRUCT
             | IMPRESION_CONSOLA
             | FUNCIONES

STRUCT → "struct" VARIABLE "{" ( TIPO_VARIABLE VARIABLE ";" )* "}"

IMPRESION_CONSOLA → "print" "(" INTERPOLACION ")" ";"

INTERPOLACION → "\"" CONTENIDO_INTERPOLADO "\""
```

CONTENIDO_INTERPOLADO → (TEXTO_LITERAL | TEXTO_INTERPOLADO)*

TEXTO_LITERAL → [^"$"]+

TEXTO_INTERPOLADO → "$" "{" VARIABLE "}"

FUNCIONES → TIPO_VARIABLE VARIABLE "(" (PARAMETROS)* ")" "{"
CONTENIDO_FUNCION* "return" VARIABLE ";" "}"
| "void" VARIABLE "(" (PARAMETROS)* ")" "{" CONTENIDO_FUNCION* "}"

CONTENIDO_FUNCION → DECLARACION_STRUCT
| IMPRESION_CONSOLA
| ASIGNACION_VARIABLES
| DECLARACION_VARIABLES
| CICLO_FOR

DECLARACION_STRUCT → TIPO_VARIABLE VARIABLE ";"

PARAMETROS → TIPO_VARIABLE VARIABLE "*" ? ("," TIPO_VARIABLE VARIABLE "*"?)*

CONDICIONAL → "if" "(" CONDICION ")" "{" BLOQUE_CONDICIONAL* "}" ELSE_PARTE?

ELSE_PARTE → "else" condicional
| "else" "{" BLOQUE_CONDICIONAL* "}"

CONDICION → EXPRESION (">" | "<" | ">=" | "<=" | "==" | "!=") EXPRESION

BLOQUE_CONDICIONAL → DECLARACION_VARIABLES
| ASIGNACION_VARIABLES
| CONDICIONAL
| IMPRESION_CONSOLA

CICLO_FOR → "for" "(" "int" "i" "=" NUMERO ";" CONDICION ";" ("i++" | "i--") ")" "{"
BLOQUE_CONDICIONAL* "}"

NUMERO → [0-9]+ ("." [0-9]+)?

CADENA → "" txt:[^"\n\r"]* ""

CARACTER → "'" [a-zA-Z0-9] "'"

BOOLEAN → ("true" | "false")

VARIABLE → [a-zA-Z_][a-zA-Z0-9_]*