Objetivos generales

- Familiarizar al estudiante con la herramienta JFlex
- Familiarizar al estudiante con la herramienta CUP
- Aplicar conocimientos de análisis léxico y sintáctico.

Objetivos específicos

- Que el estudiante comprenda a grandes rasgos el funcionamiento de la tabla de símbolos.
- Que el estudiante comprenda la realización de acciones básicas en la fase de análisis sintáctico.
- Que el estudiante comprenda el manejo básico de la tabla de símbolos.
- Que el estudiante implemente soluciones con una arquitectura cliente-servidor.
- Que el estudiante desarrolle habilidades en la creación de gramáticas libres del contexto.

Descripción de la actividad

Como parte del avance en la escuela de sistemas en el centro universitario de occidente, se pretende crear una herramienta que le permita al tutor académico el análisis del código fuente de los diferentes proyectos desarrollados por los estudiantes, esto buscando una agilización y eficiencia en el proceso de búsqueda de copias, dado a que existen clases con una población mayor o igual a 50 estudiantes.

Se pretende crear un plan piloto para el curso de Org. de Lenguajes y Compiladores 1, utilizando un analizador de archivos java basado en el método frecuencia de aparición de un término, el cual es la suma de todas las ocurrencias o el número de veces que aparece un término en un documento.

Por lo que es necesario llevar a cabo el análisis y posteriormente crear un reporte por cada análisis realizado, esto servirá al tutor para demostrar la transparencia en el proceso. Por cuestiones de tiempo y fin del curso, el plan piloto hará la comparación únicamente entre 2 proyectos a la vez.

Existirá una aplicación de usuario en java a la cual accede el tutor académico, subirá 2 proyectos para compararlos. La aplicación envía la información por sockets al servidor de análisis de java. El servidor de análisis realizará el proceso de detección de frecuencia de términos la cual devolverá como resultado un texto en formato json, con el resumen del análisis.

Al recibir el resultado en la aplicación cliente, donde se subieron los proyectos, el tutor académico podrá realizar un reporte de los datos del análisis y generar una vista html para la visualización de un reporte.

Servidor de Análisis

Se utilizará Java con flex y cup.

En esta aplicación se analizará el código java y tendrá una consola donde se reportan los errores (misma estructura de la práctica 1) o eventos del servidor de análisis, el lenguaje java a analizar contará con las siguientes instrucciones y la sintaxis ya conocidas de este lenguaje.

1. Imports

Descripción	**Ejemplo**
Importar Clases Java o librerías	Import java.util.* ; Import analsis.AST ;

Tabla No. 1

2. Clases

Descripción	**Ejemplo**

Clases java simples.	Visibilidad class clase_ejemplo{
	//Sentencias
	}

Tabla No. 2

3. Métodos y Funciones

Descripción	**Ejemplo**

Declaración de métodos y funciones.	Visibilidad tipo
	método_funcion(con_o_sin_parametros){
	//sentencias
	}

Tabla No. 3

4. Declaración de variables

Descripción	**Ejemplo**
Declaración de variables.	Visibilidad tipo Lista_de_variables ;

Tabla No. 4

5. Asignaciones

Descripción	Ejemplo
La asignación puede ser al momento de declarar o después de declarar.	Visibilidad tipo Lista_de_variables = expresión; O Lista_de_variables = expresión ;

Tabla No. 5

6. Tipos de datos:

Descripción	Ejemplo
Los tipos de datos a utilizar son.	Enteros = int Booleanos = boolean Cadenas = String Carácter = char Decimales = double Objetos = Object Donde los objetos son instancias de otras clases.

Tabla No. 6

7. Sentencias de control.

Descripción	Ejemplo
Las sentencias de control son son.	If ,if else, for, while, do while, switch.

Tabla No. 7

8. Operadores aritméticos

Descripción	Ejemplo
Las operadores posibles son	Relacionales, lógicos, incremento/decremento, agrupación.

Tabla No. 8

9. Comentarios

Descripción	Ejemplo
Los dos tipos de comentarios, simples y multilínea	//este es un comentario simple. /*este Es multi línea */

Tabla No. 9

10. Constructor

Descripción	Ejemplo

La clase puede tener su constructor donde se inicializan las variables	Public
	Nombre_clase(Con_o_Sin_Parámetros){
	//sentencias
	}

Tabla No. 10

11.Visibilidad de variables y clases:

Descripción	Ejemplo
La visibilidad de las clases y variables.	Private, public, protected, final

Tabla No. 11

12. Sentencias de salida.

Descripción	Ejemplo
Las dos sentencias de flujo que pueden vernir dentro de un while, for, switch, y funciones.	Break , Return

Tabla No. 12

13. Llamadas a funciones.

Descripción	Ejemplo
Llamadas a métodos y funciones com:	Saludar(con_parametros); Parse(sin_parametros);

Tabla No. 13

IMPORTANTE: Se omitirán vectores y matrices, solo es válido el uso de variables sencillas.

Ejemplo de un archivo simple

```
import java.util.*; // comentario simple
import java.sql.base1; // Statement,
public class ejemplo_final
{
        private String value; // "Comentarios"
        private boolean valid;
/** * Comentariooo * multilinea*/
public boolean parse() {
         Objeto nuevo_objeto = new Object();
         Saludar("hola "+" numero: "+ (55+55));
        }
public String Saludar(String val) {
                lf(val!=""){
                     return val; // "Hola aqui retorno"}
                 }
        }
}
```

IMPORTANTE: Este ejemplo no será usado como parte de la calificación y es solo una representación final por lo que al hacer sus pruebas se recomienda usar archivos más extensos.

Análisis a realizar en el servidor

1. Se analizarán los archivos de cada proyecto buscando:

· Variables del mismo tipo y nombre repetidos

· Clases del mismo nombre.

· Métodos y funciones con el mismo nombre, tipo y cantidad de parámetros

· Comparación de comentarios línea y multilínea

2. Se generará un json de respuesta que será interpretado en la aplicación de usuario y en este se incluirán:

Score.

Clases repetidas entre proyectos.

- Nombre Clase

Variables repetidas entre proyectos.

- Nombre variable
- Tipo variable
- Nombre de la Función padre

Métodos / Funciones repetidos entre proyectos.

- Nombre
- Tipo
- Número de Parámetros

Comentarios repetidos entre proyectos.

- Texto de comentario

3. El Score resultante se calculará de la siguiente forma:

Comentarios repetidos/ (Sumatoria comentarios proyecto 1 y 2))*0.25 +

Variables repetidas / (sumatoria variables proyecto 1 y 2) * 0.25 +

Métodos repetidos / (sumatoria métodos proyecto 1 y 2)* 0.25+

Clases repetidas / (sumatoria clases en proyecto 1 y 2)* 0.25

Importante: El sistema debe ser lo suficientemente inteligente, para contar las variables repetidas una única vez por lo tanto si existieran al mismo tiempo 2 variables que tienen el mismo nombre en cada proyecto deben ser tomadas una única vez individualmente

Ejemplo:

```
public class Lampara {
    public String color;
    private String nombre;
        public void cambio(String color){
        this.color=color;
        }

}
```

```
public class Lampara1 {
    private String color;
    private String nombre1;
        public void cambio1(String color){
        this.color=color;
        }

}
```

Coincidencias:

Nombre:color Tipo: String, Funcion:”Clase Lampara, Clase Lampara1, Metodo cambio1”

Nombre:color Tipo: String, Funcion:”Metodo cambio, Clase Lampara1, Metodo cambio1”

El score seria

((4)/6)*0.25= 0.167

Criterios de Repitencia

Se considera repetido algún componente cuando se cumple alguna de las siguientes condiciones.

Clase repetida

- Mismo identificador
- Mismos métodos/funciones (por su nombre)

Variable repetida

- Mismo identificador y tipo

Método/Función repetida

- Mismo nombre
- Mismos parámetros (variables, nombre y tipo)

Comentario repetido

- Mismo texto

Archivo Json generado del análisis

```
{
        Score: "0.75" ,
        Clases:[
                { Nombre: "clase1"}, {Nombre:"clase2"}
            ],
        Variables:[
                {Nombre: "var1", Tipo:"int", Funcion: "funcion1, funcion2"},
                {Nombre: "var2", Tipo:"int", Funcion: "funcion2, Clase hola"}
            ],
        Metodos:[
                {Nombre: "metodo1",Tipo:"void" , Paremetros: 2},
                {Nombre: "metodo2",Tipo:"String",  Parametros:0}
            ],
        Comentarios:[
                { Texto: "hola es un comentario"},
                { Texto: "otro coment"}
            ]
}
```

Para la parte de función en las variables se debe guardar el metodo o funcion en donde se definió la variable, si es un atributo de la clase se debe guardar como función el nombre de la clase: ejemplo

Proyecto 1

```
public class var{
        int Variable1;
}
```

Proyecto 2

```
public void ejecutar(int Variable1){
//Codigo aqui
}
```

```
Variable:[

        {Nombre: “Variable1”, “Tipo”: “Int”, Funcion:”Clase var,  Método ejecutar”}

]
```

Aplicación de usuario

Esta será una aplicación de escritorio tradicional en la cual el tutor académico podrá cargar dos proyectos (Carpeta con 0 o más archivos .java) y podrá generar reportes a partir de del análisis del proyecto esto con una sintaxis básica muy similar a html.

Esta aplicación se comunicará con el servidor de análisis por medio de sockets para el intercambio de información.

La aplicación deberá tener la funcionalidad para:

Comparación:

Lo cual nos permite seleccionar 2 directorios en los cuales almacenamos nuestras clases .java, una vez analizados los archivos y si estos no tienen ningún error lexico, sintactico y semantico se le solicitará al usuario el ingreso de una ruta en la cual se guardará el proyecto dicho proyecto debe almacenar el archivo json resultante, un archivo reportes.def que es donde definiremos nuestra sintaxis html la cual nos permitirá generar reportes y por ultimo un archivo con la extensión .copy el cual guarda la estructura de nuestro proyecto para ser abierto posteriormente en la aplicación de nuevo.

Abrir proyecto:

Esta funcionalidad nos permitirá abrir un archivo con extensión .copy el cual nos desplegará un proyecto generado con anterioridad.

Podremos editar tanto el fichero .def como el archivo json que fue generado del análisis.

Guardar:

Nos permite guardar cualquier cambio realizado en nuestro proyecto.

Crear reporte:

Ya sea abriendo un proyecto o analizando en ese momento dos proyectos java; en la Pestaña que hace referencia al archivo reportes.def podremos definir la estructura de nuestro reporte con lenguaje html; debe haber un botón de Analizar el cual nos permitirá validar nuestra sintaxis y cuando el análisis sea correcto se generará el reporte que se haya definido de lo contrario se tendrá una consola en la cual se mostraran los errores que se generen en el análisis.

Definición del lenguaje copy

Para los reportes se tendrán dos secciones una para la declaración de variable que serán usadas dentro del reporte; y una más para la sintaxis html para la definición de nuestro reportes.

Sección de definición de variables

Variable Global Json

El reporte JSON generado en la comparación de los proyectos es una variable global que podrá que al igual que las variables que se declaran podrá ser alcanzable tanto desde la sección de de declaración de variables como la sección del html. Este acceso a la información generada por el archivo JSON podrá ser accedida de la siguiente forma.

Accesso	Tipo
RESULT.Score	String
RESULT.Clases	Lista de objetos Clase
RESULT.Clases[0]	Objeto Clase en la posición 0 del arreglo
RESULT.Clases[0].Nombre	String
RESULT.Variables	Lista de objetos Variable
RESULT.Variables[0]	Objeto Variable en la posición 0 del arreglo
RESULT.Variables[0].Nombre	String
RESULT.Variables[0].Tipo	String
RESULT.Variables[0].Funcion	String
RESULT.Metodos	Lista de objetos Método
RESULT.Metodos[0]	Objeto Método en la posición 0 del arreglo
RESULT.Metodos[0].Nombre	String
RESULT.Metodos[0].Tipo	String
RESULT.Metodos[0].Parametros	Integer
RESULT.Comentarios	Lista de objetos Comentario

RESULT.Comentarios[0]	Objeto Comentario en la posición 0 del arreglo
RESULT.Metodos[0].Texto	String

Tabla No. 14

Variables definidas para el lenguaje

Los tipos de variables que podran venir definidas en nuestra sección de definición de variables son:

Tipo	**Entrada**
Integer	Número entero: 5,45,948, etc.
String	Cadena: “hola es una cadena”

Tabla No. 15

Las variables serán declarables de la siguiente forma, las opciones dentro de corchetes son opcionales.

<nombre_del_tipo_de_dato> <nombre_de_la_variable> [= <expresión>];

<nombre_del_tipo_de_dato> <nombre_de_la_variable_0>, <nombre_de_la_variable_1>,<nombre_de_la_variable_n>;

Ejemplos:

Integer var1;
Integer var1 = 5+6*5;
Integer var1, var2, var3;
String hola=”Hola ”+ “Mundo”

Asignación de valores:

A las variables definidas se les podrá asignar o cambiar su valor mediante la sintaxis:

<nombre_de_la_variable> = <expresión> ;

Ejemplo

entero1 = 1;
cadena1=RESULT.Metodos[entero1].Nombre;
cadena2="hola "+"Resultado"
cadena3=RESULT.Score;

Casteo automático de variables:

Los resultados de tipos para las operaciones con diferente tipo en cualquier orden son las siguientes. Para los operandos con tipos iguales se mantiene el tipo.Esto servirá a la hora de asignar un valor y comparar tipos a la variable a asignar.

Operación	Operandos	Resultado	Aplicación
suma	String+ Integer	String	"string"+5
resta	Integer– Integer	Integer	5-4
división	Integer / Integer	Integer	5/2
multiplicación	Integer*Integer	Integer	6*5

Tabla No. 16

Operadores Aritméticos reconocidos:

Operador	Operación	Aplicación
+	Suma	5+var2
-	Resta	Var3-var2
/	división	Var3/var2
*	multiplicación	Var3*var6*5.5
()	Agrupación	(5+5)*2/(8-7)

Tabla No. 17

IMPORTANTE: LA SECCIÓN DE DEFINICIÓN DE VARIABLES INICIA EN EL PRINCIPIO Y TERMINA AL ENCONTRAR LA ETIQUETA

<html>

Comentarios:

Los comentarios podrán ser definidos tanto en en la sección de definición de variables como en la sección del html. y tendrán la siguiente estructura:

(</) comentario (/>)

Ejemplo:

</ Este es un comentario
multilínea />

Sección de definición de html

Definición de etiquetas permitidas:

Se utilizará una simplificación de html y las etiquetas que se pueden definir son:

Etiqueta	Tipo, Función
<html></html>	Inicio y finalización del html
	String, Tamaño de letra
	String, Tamaño de letra 2
<table></table>	Tabla, Inicio de una taba
<for iterador:i hasta:Entero1;> </for>	Iterador, repetir una secuencia desde un punto hasta otro
<tr> </tr>	Fila Columna, Indica desde donde a donde abarca una fila de la columna
<th></th>	Columna título, crea una columna del tipo título para una tabla
<td> </td>	Columna datos, crea una columna de tipo datos para una tabla.
 	Espacio, deja un enter de espacio entre los componentes.

Tabla No. 18

Descripción de la etiqueta for:

Esta etiqueta hace uso de dos variables de tipo Integer declaradas en el código, la primer (dada por iterador:VARIABLE) es estrictamente una variable definida en la sección de definición de variables la cual irá cambiando su valor en 1 en cada iteración del ciclo, y la segunda (Dada por hasta:Numérico) es estrictamente una variable numérica la cual nos indica hasta qué cantidad podría llegar nuestra variable iterador.

Acceso a variables en la sintaxis html:

Dentro de cualquier etiqueta podremos escribir cualquier texto pero también podremos acceder a los valores dentro de cualquier variable para poder visualizarla de la siguiente forma:

$$ (<Nombre_Variable>) $$

Ejemplo

<h1>El score es: $$(RESULT.Score)$$ </h2>

Ejemplo completo

```
</ iniciare a definir  de alguna manera />
Integer Max, i;
Max=4;
i=0;
String texto=”Su score fue de: ”+RESULT.Score

</ Aqui defino el html />
<html>
        <h1>$$( texto )$$<h1>
        <table>
                <tr>
                        <th>Numero<th>
                        <th>Variable <th>
                        <th>Tipo <th>
                        <th>Función <th>
                 </tr>
                <for iterador:i hasta:Max;>
                        <tr>
                                <td> $$( i )$$<td>
                                <td> $$( RESULT.Variables[i].Nombre )$$<td>
                                <td> $$( RESULT.Variables[i].Tipo )$$ <td>
                                <td> $$( RESULT.Variables[i].Funcion )$$ <td>
                        </tr>
                 </for>
        </table>
</html>
```

Nota: Los lenguajes de la aplicación de usuario NO son Case sensitive, es decir, html y HtmL significan lo mismo.

Salida:

Su score fue de: 0.34

Numero	Variable	Tipo	Funcion
0	i	int	Iterado
1	camino	Objeto	Crear_camino
2	operar	Double	Suma
3	nombre	String	Clase usuario, método ejecutar
4	imprimir	String	Clase usuario, Clase usuario, método ejecutar

Más información sobre las etiquetas:

1. Las etiquetas <tr> </tr> solo pueden ser definidas dentro de las etiquetas <table> </table>
2. Las etiquetas <th></th> solo pueden estar definidas dentro de las etiquetas <tr> </tr>
3. Las etiquetas <td> <td> solo pueden estar definidas dentro de las etiquetas <tr> </tr>
4. Las etiquetas <html> </html> solo pueden estar definidas una vez.
5. Las etiquetas
, <h1></h1>, <h2> </h2> pueden venir en cualquier lugar o dentro de cualquier otra etiqueta.

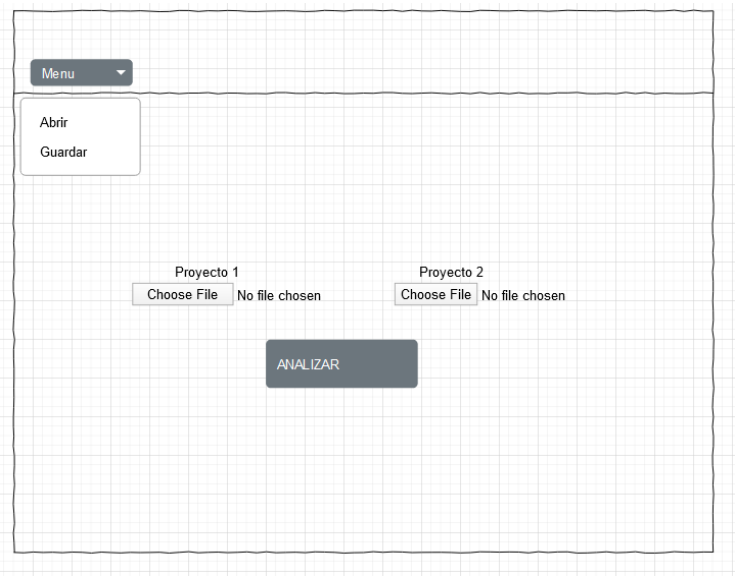
Consola de reporte de errores:

La aplicación de usuario deberá contar con una consola en la cual se deben ver reflejados los errores con los mismos atributos vistos en la Práctica 1 en caso de que el análisis falle.

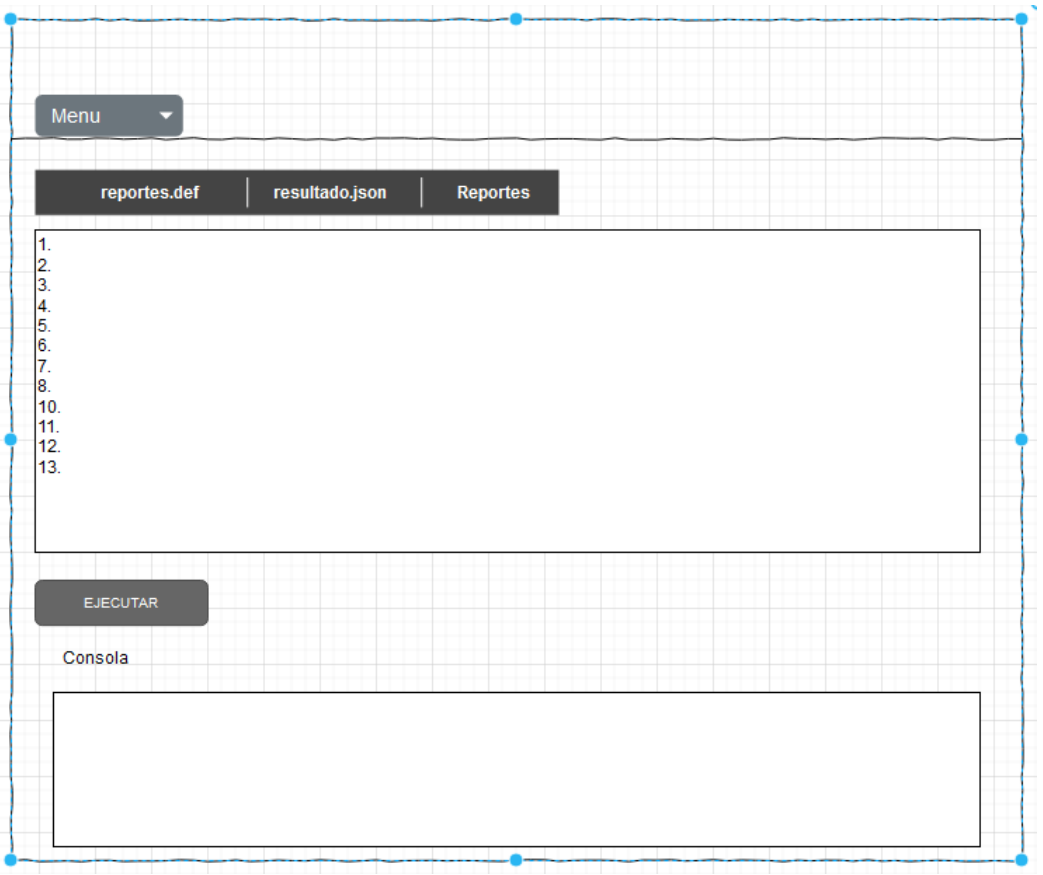
Interfaz gráfica sugerida:

La interfaz gráfica propuesta es solo una idea de cómo podría hacerse, pero ustedes pueden tomar otros criterios siempre y cuando la funcionalidad sea la misma

APP CLIENTE:



Proyecto Analizado o cargado:

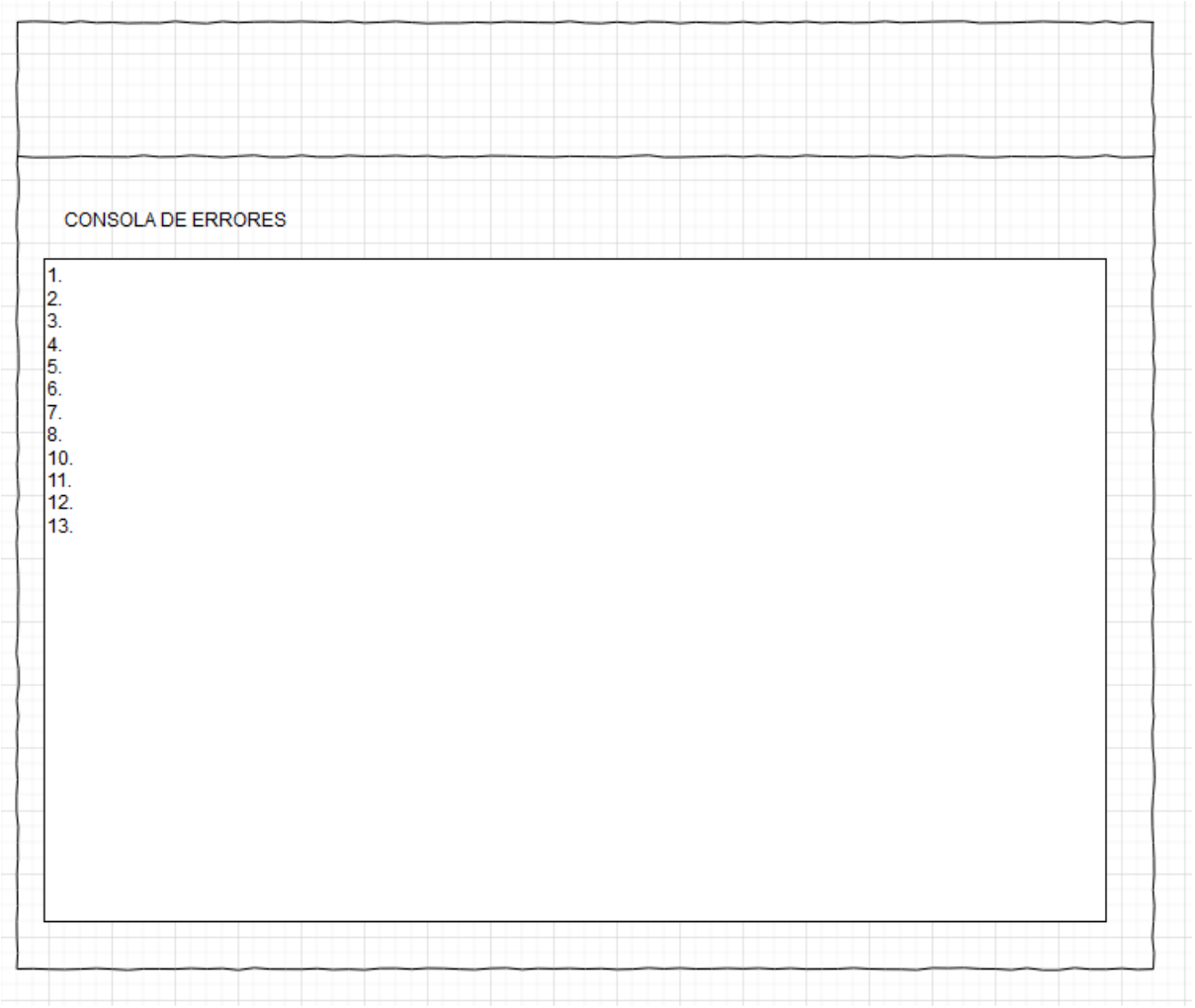


Pestaña reportes.def: Donde podremos ver y editar el archivo con el que generamos nuestro reporte.

Pestaña resultado.json: Donde podremos ver y editar el archivo JSON que se genere de nuestro análisis.

Pestaña Reportes: Aquí podremos ver gráficamente el html que definimos en la sección de reportes.def

SERVIDOR:



Arquitectura de la aplicación

Será una arquitectura cliente servidor en donde para el análisis de los dos proyectos java La aplicación de usuario y el servidor se comunicarán por medio de sockets.

Las dos aplicaciones deben ser creadas con el lenguaje java y como aplicaciones comunes de escritorio.

Importante

- Toda sección de edición de archivos debe tener una numeración de línea para poder detectar los errores con mayor facilidad.
- El proyecto debe ser desarrollado con el lenguaje java y como una aplicación de escritorio típica.
- Usar herramientas Jflex y Cup para cualquier tipo de análisis/proceso léxico y sintáctico.
- Proyecto obligatorio para tener derecho al siguiente proyecto.
- Las copias obtendrán nota de cero y se notificará a coordinación.
- Si se va a utilizar código de internet, entender la funcionalidad para que se tome como válido.
- Para recuperar el archivo json como un objeto es importante usar tanto jflex y cup para la creación.

Entrega

La fecha de entrega es el día de 06 Abril a las 14:00 horas Los componentes a entregar utilizando un repositorio git son:

- Código fuente
- Archivo jar
- Manual técnico: Detalle de la organización de su proyecto, análisis de gramática para analizador léxico y gramática para analizador sintáctico, diagrama de clases.
- Manual de usuario.

Calificación

Pendiente de definir.