Manual Técnico App Forms

Herramientas de Desarrollo

Las herramientas (ide's, lenguaje de programación, servidor web, librerías, etc) usadas para el desarrollo de las aplicaciones web y cliente(desktop), se resumen en el siguiente listado:

- Lenguaje: Java 8
  - Java Version: 1.8.0_251
- Analizador Léxico: JFlex
  - Version: 1.8.2
  - Sitio de descarga: https://jflex.de/download.html
- Analizador Sintáctico: JCup
  - Versión: 0.11b
  - Sitio de descarga: http://www2.cs.tum.edu/projects/cup/
- Implementación de Web Service JAX-RS: Jersey
  - Version: 2.33
  - Documentación: https://eclipse-ee4j.github.io/jersey.github.io/documentation/latest/index.html
- Framework front-end: Bootstrap v5.0.0-beta3
  - Sitio Web: https://getbootstrap.com/
- IDE: Apache NetBeans IDE 12
- Servidor de aplicaciones: Glassfish 5.1.0
  - Sitio de descarga: https://www.eclipse.org/downloads/download.php?file=/glassfish/glassfish-5.1.0.zip

Sistema Operativo

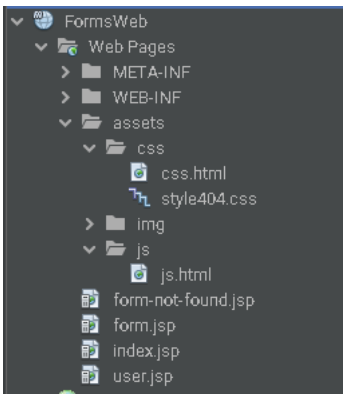
La aplicación web y cliente, fueron desarrolladas bajo el siguiente sistema operativo:

- OS: Arch Linux
- Version Kernel: x86_64 Linux 5.11.1-arch1-1

Organización del Código Fuente de la aplicación Web

La aplicación web, que como ya se indicó fue desarrollada en el lenguaje Java, presenta la siguiente estructura.

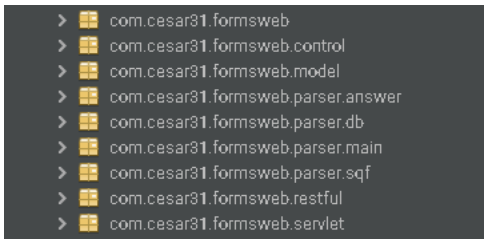
La parte web del proyecto, FormsWeb(nombre que recibe la aplicación web) sigue la siguiente estructura:



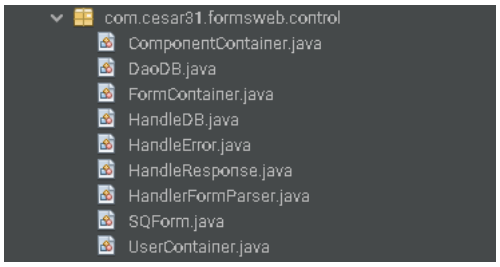
Se manejan básicamente 4 páginas web:

- **index.jsp:** se tiene la vista de inicio de sesión y búsqueda de formularios.
- **user.jsp:** Se tiene la vista o perfil del usuario con un resumen de sus formularios.
- **form.jsp:** Se tiene la vista de los formularios creados, dentro existe un método para renderizar los formularios.
- **form-not-found.jsp:** muestra la vista para algún error cuando el usuario busque o ingrese a una URL de un formulario que no existe o ha sido eliminado.

Para la parte lógica de la aplicación web o backend, se maneja la siguiente distribución de packages.

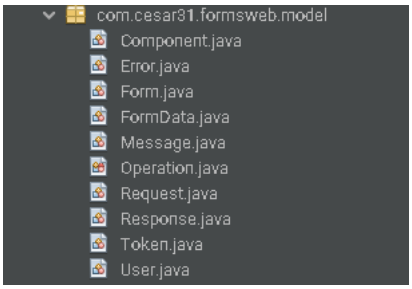


package com.cesar31.formsweb.control



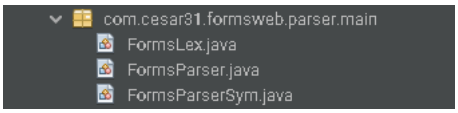
En este paquete están las clases encargadas de la parte lógica de la aplicación, el acceso a la base de datos y también los elementos encargados de crear los objetos (POJOS) necesarios para el funcionamiento de la misma, también la clase encargada de crear las respuestas que se envían hacia la aplicación cliente y las instancias de las clases destinas al análisis léxico y sintáctico de las instrucciones recibidas por el cliente y también de las estructuras de los archivos que tienen como función, almacenar los datos de usuarios, formularios y datos recolectados por formularios.

package com.cesar31.formsweb.model



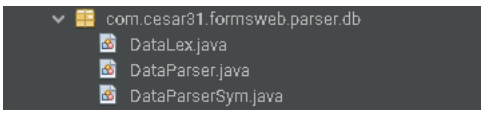
Acá se tienen almacenadas los POJOS, necesarios para el funcionamiento de la aplicación, observe que se tienen clases para crear formularios, componentes, usuarios, tokens, peticiones, respuestas y errores, estas últimas dos, se utilizan para crear la estructura de los mensajes enviados hacia la aplicación cliente.

package com.cesar31.formsweb.parser.main



En este paquete se tienen almacenadas las clases usadas para el análisis léxico y sintáctico de las peticiones de la aplicación cliente, las expresiones regulares y gramáticas usadas para generar estas clases se explican más adelante. Cabe mencionar que estas clases se generaron con las herramientas JFlex y JCup ya descritas anteriormente.

package com.cesar31.formsweb.parser.db

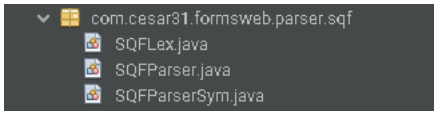


Aquí se almacenan las clases destinadas para el análisis léxico y sintáctico del archivo usado para almacenar la información de la estructura de los formularios y sus componentes y de la información de los usuarios. Más adelante se presenta la estructura del archivo de almacenamiento.

package com.cesar31.formsweb.parser.answer

Paquete destinado para guardar las clases que se utilizan para el análisis léxico y sintáctico del archivo encargado de almacenar las respuestas recolectadas por los formularios, la gramática, expresiones regulares y estructura del archivo se explican más adelante.

package com.cesar31.formsweb.parser.sqf

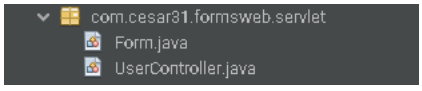


Paquete usado para las clases para el parseo de las consultas que se pueden hacer desde la aplicación cliente.

package com.cesar31.formsweb.restful

Este paquete almacena la clase usada para la comunicación del servidor con la aplicación cliente, su objetivo es recibir los mensajes o peticiones y enviarlas a las clases encargadas de hacer él parseó para posteriormente responder al cliente con las repuestas o errores encontradas en las peticiones.

package com.cesar31.formsweb.servlet



En este paquete están las clases o servlets usados para las distintas peticiones que el usuario pueda hacer mediante un navegador web y las repuestas que el servidor pueda brindar.

Análisis Léxico de las peticiones de la Aplicación Cliente (Expresiones Regulares)

La parte del análisis léxico de las peticiones que la aplicación cliente puede realizar, están a cargo de la clase FormsLex.java que como ya se describió, fue generada por la herramienta JFlex, la descripción de las expresiones regulares usadas es la siguiente:

Saltos de línea y espacios en blanco, es necesario declarar tales expresiones regular para poder incluirlas en algunas expresiones regulares que así lo requieran y para poder ignorarlas en otras y que no hagan match como errores.

```
LineTerminator = \r|\n|\r\n
WhiteSpace = {LineTerminator}|[\s\t\f]
```

Expresiones regulares para enteros y para las fechas, nótese que las fechas deberán seguir el formato yyyy-mm-dd.

```
Integer =  0|[1-9][0-9]*
Date = \d{4,4}\-\d{2,2}\-\d{2,2}
```

Símbolos y parámetros que no son parte del lenguaje de las peticiones, pero cuyas ocurrencias como cadenas pueden ser tomadas como errores léxicos o sintácticos.

```
Param = \w+
Symbol = ([\\\*\.\(\)\?\^&@¡#%;¿] | {Param})+
```

Input hace referencias a todos los caracteres exceptuando saldos de línea, enter's comillas, diagonales y pleca o barra vertical, InputClean por su lado exceptúa también espacios, tabulaciones, saltos de página.

InputCleanId se utiliza para los nombres de los nombres de los formularios y para complementar la expresión regular de Id, usada como su nombre lo indica para los id's de formularios, o componentes.

Nótese que InputCleanId elimina caracteres que se pueden interpretar como otros caracteres que son importantes en la gramática(mayor que, menor que, llaves, corchetes, comas, comillas, apostrofes).

```
Input = [^\n\r\"\\\|]+
InputClean = [^\n\r\"\\\t\s\f\|]+
InputCleanId = [^\n\r\"\\\t\s\f><\{\}\[\]=\*\+\,\|\?\'\’]+
Id = \" [\_\-\$] ([\_\-\$] | {InputCleanId} )+ \"
Name = \" {InputCleanId} \"
```

str y StringNoClean hace match con aquellas cadenas que además de estar entre comillas, pueden tener o no espacios al inicio y final, pero deben de llevar al menos un carácter seguido o no de espacios, y únicamente acepta saltos de línea justo después o antes de las comillas.

str_space y StringSpace se usan para encontrar aquellas cadenas que pudiesen tener saltos de línea, espacios, además de cualquier otro carácter entre comillas, se utiliza para aquellas expresiones que aceptan espacios y/o saltos de línea

```
str = {WhiteSpace}* {InputClean}+ {Input}+ {WhiteSpace}*
str_space = ({WhiteSpace} | {Input})+

StringNoClean = \" {str} \"
StringSpace = \" {str_space} \"
```

La expresión regular para options se utiliza para hacer match con el parámetro OPCIONES de componentes como RADIO, COMBO, CHECKBOX.

```
Options = \" ({str} \| )* {str} \"
```

La ER se utiliza para hacer match con las cadenas de tipo: "CONSULTA -1", utilizadas para las solicitudes de reportes de la aplicación cliente hacia el servidor.

```
Consulta = "CONSULTA-" {Integer}
```

Para las expresiones que van entre comillas(empiezan y terminan) se usan las siguientes expresiones regulares en combinación con otras expresiones que veremos a continuación.

```
Quote = \"
Ql = {Quote} {WhiteSpace}*
Qr = {WhiteSpace}* {Quote}
```

Para el inicio y fin de solicitud(ini_solicitud y fin_solicitud), se usan las siguientes expresiones, nótese que aceptan tanto mayúsculas como minúsculas.

```
Ini = [Ii][Nn][Ii]
Sol = [Ss][Oo][Ll][Ii][Cc][Ii][Tt][Uu][Dd]
Fin = [Ff][Ii][Nn]

Ini_sol = {Ini} "_" {Sol}
Fin_sol = {Fin} "_" {Sol}
Ini_m_sol = {Ini} "_" {Sol}[Ee][Ss]
Fin_m_sol = {Fin} "_" {Sol}[Ee][Ss]
```

Las siguientes expresiones, que van en combinación con otras que se expusieron anteriormente, son palabras reservadas para iniciar y finalizar solicitudes, fechas y enteros entre comillas.

```
  {Ini_sol}
  { return symbol(INIT_SOL, yytext()); }

  {Fin_sol}
  { return symbol(FIN_SOL, yytext()); }

  {Ini_m_sol}
  { return symbol(INIT_MANY_SOL, yytext()); }

  {Fin_m_sol}
  { return symbol(FIN_MANY_SOL, yytext()); }

  /* Fecha */
  {Ql} {Date} {Qr}
  { return symbol(DATE, yytext()); }

  {Ql} {Integer} {Qr}
  { return symbol(STR_NUMBER, yytext()); }
```

Nótese que las expresiones regulares expuestas ya han sido definidas anteriormente.

Para las solicitudes para crear, editar, eliminar y login de usuarios, se hacen uso de las siguientes expresiones:

```
  {Ql} "CREAR_USUARIO" {Qr}
  { return symbol(ADD_USER, yytext()); }

  {Ql} "CREDENCIALES_USUARIO" {Qr}
  { return symbol(CRED, yytext()); }

  {Ql} "USUARIO" {Qr}
  { return symbol(USER, yytext()); }

  {Ql} "PASSWORD" {Qr}
  { return symbol(PASS, yytext()); }

  {Ql} "FECHA_CREACION" {Qr}
  { return symbol(DATE_ADD, yytext()); }

  {Ql} "MODIFICAR_USUARIO" {Qr}
  { return symbol(EDIT_USER, yytext()); }

  {Ql} "ELIMINAR_USUARIO" {Qr}
  { return symbol(DEL_USER, yytext()); }

  {Ql} "LOGIN_USUARIO" {Qr}
  { return symbol(LOGIN, yytext()); }

  {Ql} "USUARIO_ANTIGUO" {Qr}
  { return symbol(OLD_USER, yytext()); }

  {Ql} "USUARIO_NUEVO" {Qr}
  { return symbol(NEW_USER, yytext()); }

  {Ql} "NUEVO_PASSWORD" {Qr}
  { return symbol(NEW_PASS, yytext()); }

  {Ql} "FECHA_MODIFICACION" {Qr}
  { return symbol(DATE_MOD, yytext()); }
```

Para las peticiones referentes a los formularios, se utilizan las siguientes expresiones regulares (se incluyen las palabras reservadas DARK y LIGHT que son temas predefinidos en la aplicación para los formularios).

```
/* Formularios */
  {Ql} "NUEVO_FORMULARIO" {Qr}
  { return symbol(NEW_FORM, yytext()); }

  {Ql} "PARAMETROS_FORMULARIO" {Qr}
  { return symbol(PARAM_F, yytext()); }

  {Ql} "ID" {Qr}
  { return symbol(ID, yytext()); }

  {Id}
  { return symbol(ID_, yytext()); }

  {Ql} "TITULO" {Qr}
  { return symbol(TITLE, yytext()); }

  {Ql} "NOMBRE" {Qr}
  { return symbol(NAME, yytext()); }

  {Ql} "TEMA" {Qr}
  { return symbol(THEME, yytext()); }

  {Ql} "DARK" {Qr}
  { return symbol(DARK, yytext()); }

  {Ql} "LIGHT" {Qr}
  { return symbol(LIGHT, yytext()); }

  {Ql} "USUARIO_CREACION" {Qr}
  { return symbol(USER_C, yytext()); }
```

```
{Ql} "ELIMINAR_FORMULARIO" {Qr}
{ return symbol(DEL_FORM, yytext()); }

{Ql} "MODIFICAR_FORMULARIO" {Qr}
{ return symbol(EDIT_FORM, yytext()); }
```

Para consultas (i.e. "CONSULTA-1") y id's (expresiones que inician con $, -, _ y siguen con cualquier carácter alfanumérico o cualquier de esos tres símbolos) de formularios y componentes

```
{Ql} {Consulta} {Qr}
{ return symbol(CONS_NAME, yytext()); }

{Id}
{ return symbol(ID_, yytext()); }
```

Las siguientes expresiones regulares se utilizan para la creación, edición y para eliminar componentes(elementos que forman parte de los formularios)

```
{Ql} "AGREGAR_COMPONENTE" {Qr}
{ return symbol(ADD_COMP, yytext()); }

{Ql} "PARAMETROS_COMPONENTE" {Qr}
{ return symbol(PARAM_C, yytext()); }

{Ql} "NOMBRE_CAMPO" {Qr}
{ return symbol(FIELD_N, yytext()); }

{Ql} "FORMULARIO" {Qr}
{ return symbol(FORM, yytext()); }

{Ql} "CLASE" {Qr}
{ return symbol(CLASS, yytext()); }

/* CLASE COMPONENTE */
{Ql} "CAMPO_TEXTO" {Qr}
{ return symbol(TEXT_FIELD, yytext()); }

{Ql} "AREA_TEXTO" {Qr}
{ return symbol(TEXT_AREA, yytext()); }

{Ql} "CHECKBOX" {Qr}
{ return symbol(CHECKBOX, yytext()); }

{Ql} "RADIO" {Qr}
{ return symbol(RADIO, yytext()); }

{Ql} "FICHERO" {Qr}
{ return symbol(FILE, yytext()); }

{Ql} "IMAGEN" {Qr}
{ return symbol(IMG, yytext()); }

{Ql} "COMBO" {Qr}
{ return symbol(COMBO, yytext()); }

{Ql} "BOTON" {Qr}
{ return symbol(BTN, yytext()); }
/* CLASE COMPONENTE */

{Ql} "INDICE" {Qr}
{ return symbol(INDEX, yytext()); }

{Ql} "TEXTO_VISIBLE" {Qr}
{ return symbol(TEXT, yytext()); }

{Ql} "ALINEACION" {Qr}
{ return symbol(ALIGN, yytext()); }

/* Opciones alineacion */
{Ql} "CENTRO" {Qr}
```

```
{ return symbol(CENTER, yytext()); }

{Ql} "IZQUIERDA" {Qr}
{ return symbol(LEFT, yytext()); }

{Ql} "DERECHA" {Qr}
{ return symbol(RIGHT, yytext()); }

{Ql} "JUSTIFICAR" {Qr}
{ return symbol(JUSTIFY, yytext()); }
/* Opciones alineacion */

{Ql} "REQUERIDO" {Qr}
{ return symbol(REQUIRED, yytext()); }

{Ql} "SI" {Qr}
{ return symbol(YES, yytext()); }

{Ql} "NO" {Qr}
{ return symbol(NO, yytext()); }

{Ql} "OPCIONES" {Qr}
{ return symbol(OPTION, yytext()); }

{Ql} "FILAS" {Qr}
{ return symbol(ROWS, yytext()); }

{Ql} "COLUMNAS" {Qr}
{ return symbol(COLUMNS, yytext()); }

{Ql} "URL" {Qr}
{ return symbol(URL, yytext()); }

{Ql} "ELIMINAR_COMPONENTE" {Qr}
{ return symbol(DEL_COMP, yytext()); }

{Ql} "MODIFICAR_COMPONENTE" {Qr}
{ return symbol(EDIT_COMP, yytext()); }
```

Para consultar datos, se tienen las siguientes palabras reservadas

```
/* CONSULTAR_DATOS */
{Ql} "CONSULTAR_DATOS" {Qr}
{ return symbol(CONS_DATA, yytext()); }

{Ql} "CONSULTAS" {Qr}
{ return symbol(CONS, yytext()); }

{Name}
{ return symbol(NAME_F, yytext()); }
/* CONSULTAR_DATOS */

/* Input con comillas */
{StringNoClean}
{ return symbol(STR, yytext()); }

/* opciones para radio y combo */
{Options}
{ return symbol(OPTION_V, yytext()); }
```

Para el caso de caracteres especiales (i.e. >, <, !, etc) se han declarado las siguientes ER.

```
":"
{ return symbol(COLON, yytext()); }

"<"
{ return symbol(SMALLER, yytext()); }

"!"
{ return symbol(EXCL, yytext()); }
```

```
  ">"
  { return symbol(GREATER, yytext()); }

  "{"
  { return symbol(LBRACE, yytext()); }

  "}"
  { return symbol(RBRACE, yytext()); }

  "["
  { return symbol(LBRACKET, yytext()); }

  "]"
  { return symbol(RBRACKET, yytext()); }

  ","
  { return symbol(COMMA, yytext()); }

  "+"
  { return symbol(PLUS, yytext()); }

  "-"
  { return symbol(MINUS, yytext()); }

  "*"
  { return symbol(TIMES, yytext()); }

  "/"
  { return symbol(DIVIDE, yytext()); }
```

Para el caso donde el match sean enteros, palabras(caracteres alfanuméricos) y símbolos que no pertenecen al lenguaje o palabras reservadas y se utilizan para el manejo de errores.

```
  {Integer}
  { return symbol(INTEGER, yytext()); }

  {Param}
  { return symbol(PARAM, yytext()); }

  {Symbol}
  { return symbol(SYMB, yytext()); }
```

Las siguientes expresiones regulares se utilizan para el caso cuando el match sea comillas("), espacios en blanco entre comillas y espacios en blanco:

```
/* Comillas */
  {Quote}
  { return symbol(QUOTE, yytext()); }

  {Qr} {WhiteSpace}* {Ql}
  { return symbol(EMPTY, yytext()); }

  {StringSpace}
  { return symbol(STR_SPACE, yytext()); }

  {Options_space}
  { return symbol(OPTION_SPACE, yytext()); }

  {WhiteSpace}
  { /* Ignore */ }
```

Para caracteres que no se hayan incluido:

```
  [^]
  {
```

```
    System.out.println("Error: < " + yytext() + " >");
    return symbol(ERROR, yytext());
    // throw new Error("Ilegal character: <" + yytext() + ">");
  }
```

Gramática de las peticiones de la Aplicación Cliente (Análisis Sintáctico)

La parte del análisis sintáctico, como ya se mencionó anteriormente, esta a cargo de la clase FormsParser.java, clase que ha sido generada con la herramienta JCup, a continuación se describe la gramática, que hace posible ejecutar las peticiones de la aplicación cliente.

La gramática se ha definido en el archivo forms.cup, mismo que se encuentra entre el código fuente de la aplicación.

Descripción de la Gramática:

Se tienen declarados los siguientes terminales, de tipo token, clase que se encuentra entre el código fuente y que básicamente guardan información como la línea, columna, valor y número de token de cada coincidencia o match que se encuentre con JFlex:

```
terminal Token INIT_SOL, FIN_SOL, INIT_MANY_SOL, FIN_MANY_SOL;
terminal Token PARAM, INTEGER, COLON, SMALLER, GREATER, LBRACE, RBRACE, LBRACKET, RBRACKET, COMMA, EXCL;
terminal Token PLUS, MINUS, TIMES, DIVIDE;
terminal Token ERROR, SYMB, QUOTE, EMPTY, STR_SPACE, OPTION_SPACE;

terminal Token STR, STR_N, DATE, ADD_USER, CRED, DATE_ADD, EDIT_USER, DEL_USER, LOGIN, OLD_USER, NEW_USER, NEW_PASS, DATE_MOD, USER, PASS;
terminal Token NEW_FORM, PARAM_F, ID, TITLE, NAME, THEME, DEL_FORM, EDIT_FORM, ADD_COMP, PARAM_C, FIELD_N, FORM, CLASS, TEXT_FIELD, TEXT_AR
terminal Token FILE, IMG, COMBO, BTN, INDEX, TEXT, ALIGN, REQUIRED, OPTION, ROWS, COLUMNS, URL, DEL_COMP, EDIT_COMP, ID_, USER_C;
terminal Token CENTER, LEFT, RIGHT, JUSTIFY, YES, NO, OPTION_V, STR_NUMBER, DARK, LIGHT;
terminal Token CONS_DATA, CONS, CONS_NAME, NAME_F;
```

También se tienen declarados los siguientes no terminales:

```
non terminal solicitude, ini_sol, fin_sol, sol, request, req_u, body, ini_many, fin_many;

non terminal us, pass, date, add_u, add, old_u, new_u, new_p, e_date;

/* formularios */
non terminal forms, id, title, name, theme, user_c;

/* componentes */
non terminal field_n, form, kind, index, text, opt, url, kind_a, req, rows, columns, comp;

/* data*/
non terminal data, body_data, make_data, cons;

non terminal Token str_param, class_o, align, required, kind_t, str, str_param_n, name_f;
```

La gramática tiene como símbolo inicial a solicitude, cuyas producciones dan paso a solicitudes múltiples o a una sola solicitud:

```
solicitude  ::= ini_sol
            | ini_many sol fin_many
            ;
```

Al explorar el no terminal ini_many, notamos que se encarga del encabezado estilo xml para solicitudes múltiples(<!ini_solicitudes>).

```
ini_many  ::= SMALLER EXCL INIT_MANY_SOL GREATER
          | error EXCL INIT_MANY_SOL GREATER
          | SMALLER error INIT_MANY_SOL GREATER
          | SMALLER EXCL error GREATER
          | SMALLER EXCL INIT_MANY_SOL error
          ;
```

Por su lado el no terminal fin_many se encarga del cierre de solicitudes múltiples estilo xml(<!fin_solicitudes>).

```
fin_many  ::= SMALLER EXCL FIN_MANY_SOL GREATER
          | error EXCL FIN_MANY_SOL GREATER
          | SMALLER error FIN_MANY_SOL GREATER
          | SMALLER EXCL error GREATER
          | SMALLER EXCL FIN_MANY_SOL error
          ;
```

El no terminal sol se encarga de generar una solicitud como mínimo o varias solicitudes.

```
sol     ::= ini_sol
          | sol ini_sol
          ;
```

ini_sol, se encarga del encabezado estilo xml para una solicitud:

```
ini_sol   ::= SMALLER:q EXCL INIT_SOL COLON request:r
          {:
            u.setResult(q, (String) r);
          :}
          | error EXCL:q INIT_SOL COLON request:r
          {: u.setResult((Token) q, (String) r); :}
          | SMALLER:q error INIT_SOL COLON request:r
          {: u.setResult(q, (String) r); :}
          | SMALLER:q EXCL error COLON request:r
          {: u.setResult(q, (String) r); :}
          | SMALLER:q EXCL INIT_SOL error request:r
          {: u.setResult(q, (String) r); :}
          ;
```

Como se puede observar, en esta parte se implementa código Java, necesario para la creación de las solicitudes y clasificación de las mismas(Usuario, componente, formulario, agregar, editar, eliminar, etc.)

Por otro lado, el no terminal request, van los nombres de los diferentes tipos de solicitud que existen, se incluye el uso de la palabra reservada RESULT para definir el tipo de petición y enviarlo al no terminal encargado de llamar a request(ini_sol).

```
/* Usuarios */
request   ::= ADD_USER GREATER req_u /* add user */
        {: RESULT = "CREAR"; :}
        | error GREATER req_u
        {: RESULT = "CREAR"; :}
        | ADD_USER error req_u
        {: RESULT = "CREAR"; :}
        | EDIT_USER GREATER req_u /* edit user */
        {: RESULT = "EDIT"; :}
        | EDIT_USER error req_u
        {: RESULT = "EDIT"; :}
```

```
        | DEL_USER GREATER req_u /* del user */
        {: RESULT = "DELETE"; :}
        // | error GREATER req_u
        // {: RESULT = "DELETE" :}
        | DEL_USER error req_u
        {: RESULT = "DELETE"; :}
        | LOGIN GREATER req_u /* login user */
        {: RESULT = "LOGIN"; :}
        | LOGIN error req_u
        {: RESULT = "LOGIN"; :}

        /* Formularios */
        | NEW_FORM GREATER forms
        {: RESULT = "NUEVO_FORMULARIO"; :} /* NUEVO_FORMULARIO */
        | error GREATER forms
        {: RESULT = "NUEVO_FORMULARIO"; :}
        | NEW_FORM error forms
        {: RESULT = "NUEVO_FORMULARIO"; :}
        | DEL_FORM GREATER forms
        {: RESULT = "ELIMINAR_FORMULARIO"; :} /* ELIMINAR_FORMULARIO */
        | DEL_FORM error forms
        {: RESULT = "ELIMINAR_FORMULARIO"; :}
        | EDIT_FORM GREATER forms
        {: RESULT = "MODIFICAR_FORMULARIO"; :} /* MODIFICAR_FORMULARIO */
        | EDIT_FORM error forms
        {: RESULT = "MODIFICAR_FORMULARIO"; :}

        /* Componentes para formulario */
        | ADD_COMP GREATER comp
        {: RESULT = "AGREGAR_COMPONENTE"; :} /* AGREGAR_COMPONENTE */
        | error GREATER comp
        {: RESULT = "AGREGAR_COMPONENTE"; :}
        | ADD_COMP error comp
        {: RESULT = "AGREGAR_COMPONENTE"; :}
        | DEL_COMP GREATER comp
        {: RESULT = "ELIMINAR_COMPONENTE"; :} /* ELIMINAR_COMPONENTE */
        | DEL_COMP error comp
        {: RESULT = "ELIMINAR_COMPONENTE"; :}
        | EDIT_COMP GREATER comp
        {: RESULT = "MODIFICAR_COMPONENTE"; :} /* MODIFICAR_COMPONENTE */
        | EDIT_COMP error comp
        {: RESULT = "MODIFICAR_COMPONENTE"; :}

        /* Consultar datos */
        | CONS_DATA GREATER data
        {: RESULT = "SQForm"; :}
        | error GREATER data
        {: RESULT = "SQForm"; :}
        | CONS_DATA error data
        {: RESULT = "SQForm"; :}
        ;
```

Según los comentarios, se puede observar el tipo de petición, por ejemplo los primeros van orientados a crear, editar o eliminar usuario, y los últimos se encargan de peticiones para hacer consultas.

Las siguientes producciones se utilizan para los diferentes tipos de peticiones:

```
/* add, del, edit user */
req_u   ::= LBRACE CRED COLON LBRACKET body
        | error CRED COLON LBRACKET body
        | LBRACE error COLON LBRACKET body
        | LBRACE CRED error LBRACKET body
        | LBRACE CRED COLON error body
        ;

/* operaciones con formularios */
forms   ::= LBRACE PARAM_F COLON LBRACKET body
        | error PARAM_F COLON LBRACKET body
        // | LBRACE error COLON LBRACKET body
        | LBRACE PARAM_F error LBRACKET body
        | LBRACE PARAM_F COLON error body
        ;

/* Operaciones con componentes para formulario */
comp    ::= LBRACE PARAM_C COLON LBRACKET body
```

```
        | error PARAM_C COLON LBRACKET body
        // | LBRACE error COLON LBRACKET body
        | LBRACE PARAM_C error LBRACKET body
        | LBRACE PARAM_C COLON error body
        ;

/* Operaciones para peticiones de reportes */
data    ::= LBRACE CONS COLON LBRACKET body_data
        | error CONS COLON LBRACKET body_data
        | LBRACE error COLON LBRACKET body_data
        | LBRACE CONS error LBRACKET body_data
        | LBRACE CONS COLON error body_data
```

Por otro lado, aparte de la producción o el no terminal data, las siguientes producciones se usan para generar peticiones o el cuerpo de las peticiones de reportes.

```
/* Cuerpo de la solicitud */
body_data ::= LBRACE make_data RBRACE RBRACKET RBRACE fin_sol
        | error make_data RBRACE RBRACKET RBRACE fin_sol
        | LBRACE make_data error RBRACKET RBRACE fin_sol
        | LBRACE make_data RBRACE error RBRACE fin_sol
        | LBRACE make_data RBRACE RBRACKET error fin_sol
        ;

/* Se encarga de recibir una o varias peticiones o consultas */
make_data ::= cons
        | make_data COMMA cons
        ;

/* Genere una consulta */
cons    ::= CONS_NAME:n COLON str:p
        {:
          u.setParamSQF(n.getValue(), p);
        :}
        | error COLON str
        | CONS_NAME error str
        | CONS_NAME COLON error
        ;
```

A continuación se describen las producciones para los parámetros necesarios para crear a un usuario, que incluyen usuario, contraseña, fecha de creación, usuario antiguo, nuevo usuario, nueva contraseña, fecha de modificación. El código Java que se incluye en tales producciones se usa para guardar los parámetros que se indican para cada uno de los campos.

```
us      ::= USER:a COLON str_param:p
        {:
          u.setNewParam(a, p, "USUARIO");
          // System.out.println("USUARIO");
        :}
        // | error COLON str_param
        | USER error str_param
        | USER COLON error
        ;

pass    ::= PASS:a COLON str_param:p
        {:
          u.setNewParam(a, p, "PASSWORD");
          // System.out.println("PASSWORD");
        :}
        // | error COLON str_param
        | PASS error str_param
        | PASS COLON error
        ;

date    ::= DATE_ADD:a COLON DATE:p
        {:
          u.setNewParam(a, p, "FECHA_CREACION");
          // System.out.println("FECHA_CREACION");
        :}
        // | error COLON DATE
        | DATE_ADD error DATE
```

```
        | DATE_ADD COLON error
        ;

old_u ::=   OLD_USER:a COLON str_param:p
        {:
          u.setNewParam(a, p, "USUARIO_ANTIGUO");
          // System.out.println("USUARIO_ANTIGUO");
        :}
        // | error COLON str_param
        | OLD_USER error str_param
        | OLD_USER COLON error
        ;

new_u ::=   NEW_USER:a COLON str_param:p
        {:
          u.setNewParam(a, p, "USUARIO_NUEVO");
          // System.out.println("USUARIO_NUEVO");
        :}
        // | error COLON str_param
        | NEW_USER error str_param
        | NEW_USER COLON error
        ;

new_p ::=   NEW_PASS:a COLON str_param:p
        {:
          u.setNewParam(a, p, "NUEVO_PASSWORD");
          // System.out.println("NUEVO_PASSWORD");
        :}
        // | error COLON str_param
        | NEW_PASS error str_param
        | NEW_PASS COLON error
        ;

e_date  ::=   DATE_MOD:a COLON DATE:p
        {:
          u.setNewParam(a, p, "FECHA_MODIFICACION");
          // System.out.println("FECHA_MODIFICACION");
        :}
        // | error COLON DATE
        | DATE_MOD error DATE
        | DATE_MOD COLON error
        ;
```

Producciones para los parámetros al momento de crear, editar o eliminar un formulario, se incluyen id, título, nombre, tema, y usuario de creación, el parámetro fecha de creación ya fue incluido en los parámetros del usuario, posteriormente se mostrará la producción que permite unir todas estas producciones.

```
id      ::= ID:a COLON ID_:p
        {: u.setNewParam(a, p, "ID"); :}
        | ID error ID_
        | ID COLON error
        ;

title   ::= TITLE:a COLON str_param:p
        {: u.setNewParam(a, p, "TITULO"); :}
        | TITLE error str_param
        | TITLE COLON error
        ;

name    ::= NAME:a COLON name_f:p
        {: u.setNewParam(a, p, "NOMBRE"); :}
        | NAME error str_param
        | NAME COLON error
        ;

theme   ::= THEME:a COLON kind_t:p
        {: u.setNewParam(a, p, "TEMA"); :}
        | THEME error kind_t
        | THEME COLON error
        ;

user_c    ::= USER_C:a COLON str_param:p
        {: u.setNewParam(a, p, "USUARIO_CREACION"); :}
        | USER_C error str_param
```

```
        | USER_C COLON error
        ;
```

Producciones para los parámetros para crear, editar, eliminar componentes, se incluyen nombre de campo, formulario, clase, indice, texto visible, requerido, opciones, filas, columnas, URL.

```
field_n   ::= FIELD_N:a COLON str_param:p
        {: u.setNewParam(a, p, "NOMBRE_CAMPO"); :}
        | FIELD_N error str_param
        | FIELD_N COLON error
        ;

form    ::= FORM:a COLON ID_:p
        {: u.setNewParam(a, p, "FORMULARIO"); :}
        | FORM error ID
        | FORM COLON error
        ;

kind    ::= CLASS:a COLON class_o:p
        {: u.setNewParam(a, p, "CLASE"); :}
        | CLASS error class_o
        | CLASS COLON error
        ;

index   ::= INDEX:a COLON STR_NUMBER:p
        {: u.setNewParam(a, p, "INDICE"); :}
        | INDEX error STR_NUMBER
        | INDEX COLON error
        ;

text    ::= TEXT:a COLON str_param:p
        {: u.setNewParam(a, p, "TEXTO_VISIBLE"); :}
        | TEXT error str_param
        | TEXT COLON error
        ;

kind_a    ::= ALIGN:a COLON align:p
        {: u.setNewParam(a, p, "ALINEACION"); :}
        | ALIGN error align
        | ALIGN COLON error
        ;

req     ::= REQUIRED:a COLON required:p
        {: u.setNewParam(a, p, "REQUERIDO"); :}
        | REQUIRED error YES
        | REQUIRED COLON error
        ;

opt     ::= OPTION:a COLON OPTION_V:p
        {: u.setNewParam(a, p, "OPCIONES"); :}
        | OPTION error OPTION_V
        | OPTION COLON error
        ;

rows    ::= ROWS:a COLON STR_NUMBER:p
        {: u.setNewParam(a, p, "FILAS"); :}
        | ROWS error STR_NUMBER
        | ROWS COLON error
        ;

columns   ::= COLUMNS:a COLON STR_NUMBER:p
        {: u.setNewParam(a, p, "COLUMNAS"); :}
        | COLUMNS error STR_NUMBER
        | COLUMNS COLON error
        ;

url     ::= URL:a COLON str_param:p
        {: u.setNewParam(a, p, "URL"); :}
        | URL error str_param
        | URL COLON error
        ;
```

A pesar un componente, formulario o usuario en cada tipo de petición no lleva todos los campos o bien no todos son requeridos en algunas peticiones, en el código fuente se encontrarán los métodos necesarios para cada petición y verificar si las peticiones traen todos los parámetros.

Ya se han mostrado las producciones para cada uno de los parámetros para crear, editar y eliminar usuarios, formularios y componentes, ahora se presenta la producción o el símbolo no terminal que permite que el orden sea cualquiera, cabe resaltar, que para verificar si en cierta petición los parámetros ingresados son los correctos o si falto o sobro alguno, se utilizan varios métodos que se encuentran en las clases UserContainer.java, FormContainer.java y ComponetContainer.java.

```
add_u   ::= us
        | pass /* usuarios */
        | date
        | old_u
        | new_u
        | new_p
        | e_date
        | id /* formularios y componentes*/
        | title
        | name
        | theme
        | user_c
        | field_n
        | form
        | kind
        | index
        | text
        | kind_a
        | req
        | opt
        | rows
        | columns
        | url
        | error
        ;

add     ::= add_u
        | add_u COMMA add
        // | add_u error add
        ;
```

El no terminal add_u, combinado con el no terminal add, se encargan que el orden no importe al momento de ingresar los parámetros y como ya se mencionó, mediante algunos métodos se verifica que la estructura o parámetros estén completos y que no sobre o falte alguno.

El no terminal add, forma parte de la siguiente producción, que básicamente es el cuerpo de cualquier petición.

```
body    ::= LBRACE add RBRACE RBRACKET RBRACE fin_sol
        | error add RBRACE RBRACKET RBRACE fin_sol
        // | LBRACE error RBRACE RBRACKET RBRACE fin_sol
        | LBRACE add error RBRACKET RBRACE fin_sol
        | LBRACE add RBRACE error RBRACE fin_sol
        | LBRACE add RBRACE RBRACKET error fin_sol
        ;
```

Las siguientes producciones forman parte de los parámetros para la construcción de formularios y componentes. Por ejemplo, la producción required, que es opcional, indica si el componente debe responderse o no en el formulario al que pertenece.

Por otro lado está la producción class_o, que determina el tipo o clase de componente, el terminal align que tiene las opciones para la alineación de los componentes.

```
/* Tipo de componente */
class_o   ::= TEXT_FIELD:p {: RESULT = p; :}
        | TEXT_AREA:p {: RESULT = p; :}
        | CHECKBOX:p {: RESULT = p; :}
        | RADIO:p {: RESULT = p; :}
        | FILE:p {: RESULT = p; :}
        | IMG:p {: RESULT = p; :}
        | COMBO:p {: RESULT = p; :}
        | BTN:p {: RESULT = p; :}
        ;

/* Alineacion */
align   ::= CENTER:p {: RESULT = p; :}
        | LEFT:p {: RESULT = p; :}
        | RIGHT:p {: RESULT = p; :}
        | JUSTIFY:p {: RESULT = p; :}
        ;

/* Requerido */
required  ::= YES:p {: RESULT = p; :}
        | NO:p {: RESULT = p; :}
        ;

/* Tipo de tema */
kind_t    ::= DARK:p {: RESULT = p; :}
        | LIGHT:p {: RESULT = p; :}
        ;

/* parametros para distintos casos, user, pass, etc */
str_param ::= STR:p {: RESULT = p; :}
        | name_f:p {: RESULT = p; :}
        ;

/* nombre del campo del componente */
name_f    ::= NAME_F:p {: RESULT = p; :}
        | str_param_n:p {: RESULT = p; :}
        ;
```

Para los diferentes parámetros, por ejemplo usuario, contraseña, nombres, también es válido usar algunas o todas las palabras reservadas que pertenecen al lenguaje para hacer peticiones, para ello se tiene la siguiente producción.

```
str_param_n ::= DATE:p {: RESULT = p; :}
        | ADD_USER:p {: RESULT = p; :}
        | CRED:p {: RESULT = p; :}
        | USER:p {: RESULT = p; :}
        | PASS:p {: RESULT = p; :}
        | DATE_ADD:p {: RESULT = p; :}
        | EDIT_USER:p {: RESULT = p; :}
        | DEL_USER:p {: RESULT = p; :}
        | LOGIN:p {: RESULT = p; :}
        | OLD_USER:p {: RESULT = p; :}
        | NEW_USER:p {: RESULT = p; :}
        | NEW_PASS:p {: RESULT = p; :}
        | DATE_MOD:p {: RESULT = p; :}
        | NEW_FORM:p {: RESULT = p; :}
        | PARAM_F:p {: RESULT = p; :}
        | ID:p {: RESULT = p; :}
        // | ID_:p {: RESULT = p; :}
        | TITLE:p {: RESULT = p; :}
        | NAME:p {: RESULT = p; :}
        | THEME:p {: RESULT = p; :}
        | USER_C:p {: RESULT = p; :}
        | DEL_FORM:p {: RESULT = p; :}
        | EDIT_FORM:p {: RESULT = p; :}
        | ADD_COMP:p {: RESULT = p; :}
        | PARAM_C:p {: RESULT = p; :}
        | FIELD_N:p {: RESULT = p; :}
        | FORM:p {: RESULT = p; :}
        | CLASS:p {: RESULT = p; :}
        | INDEX:p {: RESULT = p; :}
        | TEXT:p {: RESULT = p; :}
        | ALIGN:p {: RESULT = p; :}
        | REQUIRED:p {: RESULT = p; :}
        | OPTION:p {: RESULT = p; :}
```

```
        | ROWS:p {: RESULT = p; :}
        | COLUMNS:p {: RESULT = p; :}
        | URL:p {: RESULT = p; :}
        | DEL_COMP:p {: RESULT = p; :}
        | EDIT_COMP:p {: RESULT = p; :}
        | OPTION_V:p {: RESULT = p; :}
        | STR_NUMBER:p {: RESULT = p; :}
        | class_o:p {:RESULT = p; :}
        | align:p {: RESULT = p; :}
        | required:p {: RESULT = p; :}
        | kind_t:p {: RESULT = p; :}
        | CONS_DATA:p {: RESULT = p; :}
        | CONS:p {: RESULT = p; :}
        | CONS_NAME:p {: RESULT = p; :}
        ;
```

Y para finalizar se tiene la producción encargada del cierre de una petición. (<fin_solicitud!>)

```
/* fin solicitud */
fin_sol   ::= SMALLER FIN_SOL EXCL GREATER
        | error FIN_SOL EXCL GREATER
        | SMALLER error EXCL GREATER
        | SMALLER FIN_SOL error GREATER
        | SMALLER FIN_SOL EXCL error
        | error
        ;
```

Estructura de la Base de Datos (Archivo de Almacenamiento)

La base de datos o los datos de los clientes, formularios y sus componentes sigue una estructura JSON como se describe a continuación:

```
USERS
[ {
  "user" : "jose_12",
  "password" : "21_password",
  "cDate_user" : "2021-03-21",
  "eDate" : "2021-03-28"
}, {
  "user" : "nuevo_usuario1",
  "password" : "#pass31",
  "cDate_user" : "2021-03-23",
  "eDate" : null
}]
FORMS
[ {
  "id_form" : "$form1",
  "title" : "Formulario 1",
  "name" : "Soy_el_formulario_1",
  "theme" : "LIGHT",
  "user_creation" : "cesar_31@user",
  "cDate_form" : "2021-03-21",
  "components" : [ {
    "id_component" : "$_grupo_paises",
    "fieldName" : "Pais",
    "form_id" : "$form1",
    "kind" : "CHECKBOX",
    "index" : 1,
    "text" : "Pais de Origen:",
    "aling" : "DERECHA",
    "required" : true,
    "url" : null,
    "options" : [ "Guatemala", "Estados Unidos", "Honduras", "OTRO" ],
    "rows" : null,
    "columns" : null
  }, {
    "id_component" : "$_text_cliente",
    "fieldName" : "Cliente",
```

```
    "form_id" : "$form1",
    "kind" : "CAMPO_TEXTO",
    "index" : 2,
    "text" : "Nombre de cliente:",
    "aling" : "CENTRO",
    "required" : true,
    "url" : null,
    "options" : [ ],
    "rows" : null,
    "columns" : null
  }, {
    "id_component" : "$_text_cliente1",
    "fieldName" : "Pais",
    "form_id" : "$form1",
    "kind" : "COMBO",
    "index" : 3,
    "text" : "Pais de Origen:",
    "aling" : "CENTRO",
    "required" : true,
    "url" : null,
    "options" : [ "Guatemala", "El Salvador", "Honduras", "OTRO" ],
    "rows" : null,
    "columns" : null
  } ]
}, {
  "id_form" : "$form2",
  "title" : "Formulario para encuesta 2",
  "name" : "formulario_encuesta_2",
  "theme" : "LIGHT",
  "user_creation" : "cesar_31@user",
  "cDate_form" : "2021-03-21",
  "components" : [ {
    "id_component" : "$_text_cliente",
    "fieldName" : "Cliente",
    "form_id" : "$form2",
    "kind" : "CAMPO_TEXTO",
    "index" : 1,
    "text" : "Nombre de cliente:",
    "aling" : "CENTRO",
    "required" : true,
    "url" : null,
    "options" : [ ],
    "rows" : null,
    "columns" : null
  }, {
    "id_component" : "$_grupo_paises",
    "fieldName" : "Pais",
    "form_id" : "$form2",
    "kind" : "RADIO",
    "index" : 2,
    "text" : "Pais de Origen:",
    "aling" : "CENTRO",
    "required" : true,
    "url" : null,
    "options" : [ "Inglaterra", "Estados Unidos", "Rusia", "España" ],
    "rows" : null,
    "columns" : null
  }, {
    "id_component" : "$_text_cliente1",
    "fieldName" : "Pais",
    "form_id" : "$form2",
    "kind" : "COMBO",
    "index" : 3,
    "text" : "Pais de Origen:",
    "aling" : "CENTRO",
    "required" : true,
    "url" : null,
    "options" : [ "Guatemala", "El Salvador", "Honduras", "OTRO" ],
    "rows" : null,
    "columns" : null
  } ]
} ]
```


Como se puede observar, el único cambio que existe respecto a la estructura JSON es que se tienen dos palabras reservadas(USER y FORMS) para separar a los datos de usuarios como de clientes, también se observa que dentro e la estructura de los formularios, se tienen objetos JSON con la estructura de los componentes.

Análisis Léxico de la Estructura del archivo de Almacenamiento (Expresiones Regulares)

Las expresiones regulares necesarias para descomponer al archivo de almacenamiento en tokens, se puede resumir de la manera siguiente:

Expresiones regulares para para enteros y espacios en blanco o saltos de linea:

```
/* Integer */
Integer = 0 | [1-9][0-9]*

/* Espacios */
LineTerminator = \r|\n|\r\n
WhiteSpace = {LineTerminator}|[\s\t\f]
```

Expresiones regulares para palabras reservadas que no estan entre comillas:

```
  "USERS"
  { return symbol(USERS); }

  "FORMS"
  { return symbol(FORMS); }

  "null"
  { return symbol(NULL, String.valueOf(yytext())); }

  "true"
  { return symbol(TRUE, Boolean.valueOf(yytext())); }

  "false"
  { return symbol(FALSE, Boolean.valueOf(yytext())); }

  {Integer}
  { return symbol(INTEGER, Integer.valueOf(yytext())); }
```

Expresiones regulares para símbolos reservados

```
  /* Simbolos reservados en la estructura JSON */
  "["
  { return symbol(LBRACKET); }

  "]"
  { return symbol(RBRACKET); }

  "{"
  { return symbol(LBRACE); }

  "}"
  { return symbol(RBRACE); }

  ":"
  { return symbol(COLON); }

  ","
  { return symbol(COMMA); }
```

Para el caso de las palabras o parámetros que están o vienen entre comillas, se utilizan el estado STRING para poder hacer match con todas las expresiones que están entre comillas.

```
\"
{
  string.setLength(0);
  yybegin(STRING);
}
```

Esto significa que si en el estado YYINITIAL se mapea o encuentra unas comillas, entonces el analizador léxico pasa al estado STRING y una vez en este estado se siguen las siguientes reglas para las diferentes expresiones regulares.

```
\"
{
  yybegin(YYINITIAL);
  return setType(STR, string.toString());
}

 [^\n\r\"\\]+
{ string.append( yytext() ); }

\\t
{ string.append('\t'); }

\\n
{ string.append('\n'); }

\\r
{ string.append('\r'); }

\\\"
{ string.append('\"'); }

\\
{ string.append('\\'); }
```

Como se puede ver para cada una de las expresiones regulares en el estado STRING se agregan a string(que es un StringBuffer) hasta que se encuentran un unas de comillas, y se regresa al estado YYINITIAL.

Para el caso de palabras reservadas entre comilas, se utiliza el siguiente método de tipo symbol para devolver cada palabra reservada como un token o symbol para el análisis sintáctico, metodo que se llama desde el estado STRING cuando se encuentran las comillas de cierre para volver al estado YYINITIAL.

```
private Symbol setType(int type, Object value) {
    String s = value.toString();

    switch(s) {
      case "user":
        return symbol(USER, s);
      case "password":
        return symbol(PASS, s);
      case "cDate_user":
        return symbol(DATE_USER, s);
      case "eDate":
        return symbol(E_DATE, s);
      case "id_form":
        return symbol(ID_FORM, s);
      case "title":
        return symbol(TITLE, s);
      case "name":
        return symbol(NAME, s);
      case "theme":
        return symbol(THEME, s);
      case "user_creation":
        return symbol(USER_C, s);
      case "cDate_form":
        return symbol(DATE_FORM, s);
      case "components":
        return symbol(COMP, s);
      case "id_component":
        return symbol(ID_COMP, s);
      case "fieldName":
        return symbol(FIELD_N, s);
```

```
      case "form_id":
        return symbol(FORM_ID, s);
      case "kind":
        return symbol(TIPO, s);
      case "index":
        return symbol(INDEX, s);
      case "text":
        return symbol(TEXT, s);
      case "aling":
        return symbol(ALING, s);
      case "required":
        return symbol(REQUIRED, s);
      case "url":
        return symbol(URL, s);
      case "options":
        return symbol(OPT, s);
      case "rows":
        return symbol(ROWS, s);
      case "columns":
        return symbol(COLUMNS, s);

      default:
        return symbol(STR, s);
    }
  }
```

Como se puede observar, en el caso de que el objecto recibido no coincide con las palabras reservadas, entonces se devuelve o enviar un objeto de tipo symbol cuyo token o sym se llama STR.

Análisis Sintáctico de la Estructura del archivo de almacenamiento (Gramática)

El análisis sintáctico del archivo de almacenamiento esta a cargo del archivo DataParser.java, que como ya se mencionó fue generado con JCup. El archivo fuente con el que se generó DataParser.java se encuentra entre el código fuente del proyecto y la gramática tiene la siguiente estructura.

Los terminales y no terminales usados son los siguientes:

```
terminal FORMS, USERS, LBRACKET, RBRACKET, LBRACE, RBRACE, COLON, COMMA;
terminal ERROR;
terminal String USER, PASS, DATE_USER, E_DATE, NULL, ID_FORM, TITLE, NAME, THEME, USER_C, DATE_FORM, COMP, ID_COMP, FIELD_N;
terminal String FORM_ID, TIPO, INDEX, TEXT, ALING, REQUIRED, URL, OPT, ROWS, COLUMNS, STR;
terminal Boolean TRUE, FALSE;
terminal Integer INTEGER;

non terminal json, json_form, make_u, opt, user, users, make_op, components, comps, comp, make_c, cm;
non terminal value, u_keys, f_keys, c_keys, json_user;
non terminal make_f, forms, fm;
non terminal String str;
non terminal Boolean boolean_v;
non terminal User us;
non terminal Form form;
non terminal ArrayList<String> opts;
```

La producción inicial llamada json, genera o tiene la siguiente producción:

```
json    ::= json_user json_form
        ;
```

El no terminal json_user se encarga de las producciones para la estructura para guardar usuarios y por su lado json_form se encarga de la estructura para formularios:

```
json_user ::= USERS LBRACKET make_u RBRACKET
        ;

json_form   ::= FORMS LBRACKET make_f RBRACKET
        ;
```

Las producciones para los usuarios siguen la siguiente estructura:

```
/* Usuarios */
make_u    ::= users
        |
        ;

users   ::= user
        | users COMMA user
        ;

/* USUARIO */
user    ::= LBRACE us RBRACE
        {: dao.createUser(); :}
        ;

us      ::= u_keys
        | us COMMA u_keys
        ;

u_keys    ::= USER COLON str:p
        {: dao.setParam("user", p); :}
        | PASS COLON str:p
        {: dao.setParam("pass", p); :}
        | DATE_USER COLON str:p
        {: dao.setParam("cDate", p); :}
        | E_DATE COLON str:p
        {: dao.setParam("eDate", p); :}
        ;
```

Nótese que el orden de los parámetros realmente no importa, es decir puede venir en cualquier orden.

Ahora bien para la estructura de formularios se tiene las siguientes producciones, en donde también el orden puede ser cualquiera.

```
form    ::= LBRACE fm RBRACE
        {: dao.createForms(); :}
        ;

fm      ::= f_keys
        | fm COMMA f_keys
        ;

f_keys    ::= ID_FORM COLON str:p
        {: dao.setParam("id_form", p); :}
        | TITLE COLON str:p
        {: dao.setParam("title", p); :}
        | NAME COLON str:p
        {: dao.setParam("name", p); :}
        | THEME COLON str:p
        {: dao.setParam("theme", p); :}
        | USER_C COLON str:p
        {: dao.setParam("user_c", p); :}
        | DATE_FORM COLON str:p
        {: dao.setParam("date_form", p); :}
        | COMP COLON components /* componentes */
        ;
```

Los formularios pueden tener componentes, por tanto, las producciones para componentes que siguen la siguiente estructura, también pueden aceptar los parámetros en cualquier orden.

```
/* COMPONENTE */
comp    ::= LBRACE cm RBRACE
        {: dao.createComponents(); :}
        ;

cm      ::= c_keys
        | cm COMMA c_keys
        ;

c_keys    ::= ID_COMP COLON str:p
        {: dao.setParamC("id_comp", p); :}
        | FIELD_N COLON str:p
        {: dao.setParamC("field", p); :}
        | FORM_ID COLON str:p
        {: dao.setParamC("form_id", p); :}
        | TIPO COLON str:p
        {: dao.setParamC("kind", p); :}
        | INDEX COLON INTEGER:p
        {: dao.setParamC("index", String.valueOf(p)); :}
        | TEXT COLON str:p
        {: dao.setParamC("text", p); :}
        | ALING COLON str:p
        {: dao.setParamC("aling", p); :}
        | REQUIRED COLON boolean_v:p
        {: dao.setParamC("req", String.valueOf(p)); :}
        | URL COLON str:p
        {: dao.setParamC("url", p); :}
        | OPT COLON opt /* Opciones */
        | ROWS COLON value:p
        {: dao.setParamC("rows", String.valueOf(p)); :}
        | COLUMNS COLON value:p
        {: dao.setParamC("cols", String.valueOf(p)); :}
        ;

/* Options */
opt     ::= LBRACKET make_op RBRACKET
        ;

make_op   ::= opts
        |
        ;

opts    ::= str:p
        {: dao.setOption(p); :}
        | opts COMMA str:p
        {: dao.setOption(p); :}
        ;
```

También se ha incluido las producciones para las opciones, que recordemos en las peticiones de la aplicación cliente vienen de la forma: opcion1 | opcion2 | ... | opcion_n, la manera de almacenar estas opciones es mediante objetos: [opction1, opcion2, ..., opcion_n].

Y por último se tienen las producciones para aceptar las opciones para los diferentes parámetros, se incluyen dos producciones especiales para valores booleanos y para valores enteros o nulos./

```
str     ::= STR:p {: RESULT = p; :}
        | USER:p {: RESULT = p; :}
        | PASS:p {: RESULT = p; :}
        | DATE_USER:p {: RESULT = p; :}
        | E_DATE:p {: RESULT = p; :}
        | NULL:p {: RESULT = p; :}
        | ID_FORM:p {: RESULT = p; :}
        | TITLE:p {: RESULT = p; :}
        | NAME:p {: RESULT = p; :}
        | THEME:p {: RESULT = p; :}
        | USER_C:p {: RESULT = p; :}
        | DATE_FORM:p {: RESULT = p; :}
        | COMP:p {: RESULT = p; :}
        | ID_COMP:p {: RESULT = p; :}
        | FIELD_N:p {: RESULT = p; :}
        | FORM_ID:p {: RESULT = p; :}
        | TIPO:p {: RESULT = p; :}
        | INDEX:p {: RESULT = p; :}
```

```
        | TEXT:p {: RESULT = p; :}
        | ALING:p {: RESULT = p; :}
        | REQUIRED:p {: RESULT = p; :}
        | URL:p {: RESULT = p; :}
        | OPT:p {: RESULT = p; :}
        | ROWS:p {: RESULT = p; :}
        | COLUMNS:p {: RESULT = p; :}
        ;

boolean_v ::= TRUE:p {: RESULT = p; :}
        | FALSE:p {: RESULT = p; :}
        ;

value   ::= INTEGER:p {: RESULT = p; :}
        | NULL:p {: RESULT = p; :}
        ;
```

Se tiene también un archivo para guardar la información o los datos recolectados por los formularios, tal archivo como se puede ver, sigue la estructura json.

```
DATOS_RECOPILADOS
[ {
  "idForm" : "$$form1",
  "nameForm" : "formulario_prueba",
  "data" : [ {
    "Cliente" : "César Reginaldo"
  }, {
    "Cliente" : "César Reginaldo Tzoc"
  } ]
}, {
  "idForm" : "$$form6",
  "nameForm" : "formulario_prueba",
  "data" : [ {
    "Cliente" : "César Reginaldo Tzoc"
  }, {
    "Cliente" : "César Reginaldo Tzoc"
  } ]
}, {
  "idForm" : "$_form7",
  "nameForm" : "formulario_test_7_v2",
  "data" : [ {
    "Equipo" : "FC Barcelona",
    "Edad" : "23",
    "Identificacion" : "/home/cesar31/Java/AppForms/FormsWeb/DB/Files/error.png",
    "Pais" : "Guatemala",
    "Cliente" : "Mi nombre es nombre."
  } ]
} ]
```

Se puede ver que guarda el ID y nombre del formulario y posteriormente un objeto JS, con los datos recopilados.

Aunque la estructura es muy similar al archivo para guardar la información de usuarios y la estructura de formularios y componentes, también se tienen un parser para el archivo que almacena los datos recopilados por los formularios.

Análisis Léxico del archivo para almacenar datos recopilados por formularios

Para números enteros, espacios y saltos de linea:

```
/* Integer */
Integer = 0 | [1-9][0-9]*

/* Espacios */
LineTerminator = \r|\n|\r\n
WhiteSpace = {LineTerminator}|[\s\t\f]
```

Se manejan dos estados(YYINITIAL y STRING), las expresiones en el primer estado, que básicamente son algunas palabras reservadas que no van entre comillas y símbolos especiales o reservados.

```
"DATOS_RECOPILADOS"
  { return symbol(COLLECTED); }

  "["
  { return symbol(LBRACKET); }

  "]"
  { return symbol(RBRACKET); }

  "{"
  { return symbol(LBRACE); }

  "}"
  { return symbol(RBRACE); }

  ":"
  { return symbol(COLON); }

  ","
  { return symbol(COMMA); }

  \"
  {
    string.setLength(0);
    yybegin(STRING);
  }

  {WhiteSpace}
  { /* Ignore */ }
```

Como ya se mencionan anteriormente con el análisis léxico para el archivo para guardar usuarios y formularios, con una camilla se cambia al estado STRING, lo mismo sucede aquí:

```
<STRING> {

  \"
  {
    yybegin(YYINITIAL);
    return setType(STR, string.toString());
  }

    [^\n\r\"\\]+
  { string.append(yytext()); }

  \\t
  { string.append('\t'); }

  \\n
  { string.append('\n'); }

  \\r
  { string.append('\r'); }

  \\\"
  { string.append('\"'); }

  \\
  { string.append('\\'); }

}
```

También se tiene el siguiente método para clasificar algunas palabras que son reservadar y van entre comillas:

```
private Symbol setType(int type, Object value) {
    String s = value.toString();

    switch(s) {
      case "idForm":
        return symbol(ID, s);
```

```
      case "nameForm":
        return symbol(NAME, s);
      case "data":
        return symbol(DATA, s);
      default:
      return symbol(STR, s);
    }
  }
```

Análisis Sintáctico del archivo para datos recopilados por formularios

Se usan los siguientes terminales y no terminales:

```
terminal COLLECTED, LBRACKET, RBRACKET, LBRACE, RBRACE, COLON, COMMA;
terminal ERROR;
terminal String ID, NAME, DATA, STR;

non terminal json_data, make_d, data_m_form, data_form, form, fd_keys, data, make_data, data_fms, data_f, data_body, data_keys;
non terminal String str;
```

La producción inicial `json_data` tiene la siguiente estructura:

```
json_data   ::= COLLECTED LBRACKET make_d RBRACKET
            ;
```

Donde `make_data`, es la producción encargada de generar o aceptar datos de los formularios, también puede que no haya datos y en tal caso aceptaría "cadenas vacias".

```
make_d    ::= data_m_form
          |
          ;

data_m_form ::= data_form
            | data_m_form COMMA data_form
            ;
```

Las producciones `data_form` y `form`, es la que verifica que dentro de la estructura del archivo para almacenar información vengan los parámetros o datos necesarios de un formulario, id y nombre.

```
data_form ::= LBRACE form RBRACE
        {: dao.createFormWithData(); :}
        ;

form    ::= fd_keys
        | form COMMA fd_keys
        ;

fd_keys   ::= ID COLON str:q
        {: dao.setDataFm("ID", q); :}
        | NAME COLON str:q
        {: dao.setDataFm("NAME", q); :}
        | DATA COLON data
        ;
```

Y por último la producción data con ayuda de las otras producciones que se incluyen a continuación verifica la estructura de la información recopilada en cada formulario:

```
data    ::= LBRACKET make_data RBRACKET
        ;

make_data ::= data_fms
        |
        ;

data_fms  ::= data_f
        | data_fms COMMA data_f
        ;

data_f    ::= LBRACE data_body RBRACE
        {: dao.createDataForm(); :}
        ;

data_body ::= data_keys
        | data_body COMMA data_keys
        ;

data_keys ::= str:p COLON str:q
        {: dao.setDataParam(p, q); :}
        ;

str     ::= STR:p {: RESULT = p; :}
        | ID:p {: RESULT = p; :}
        | NAME:p {: RESULT = p; :}
        | DATA:p {: RESULT = p; :}
        ;
```

Estructura de las respuestas o mensajes del Servidor

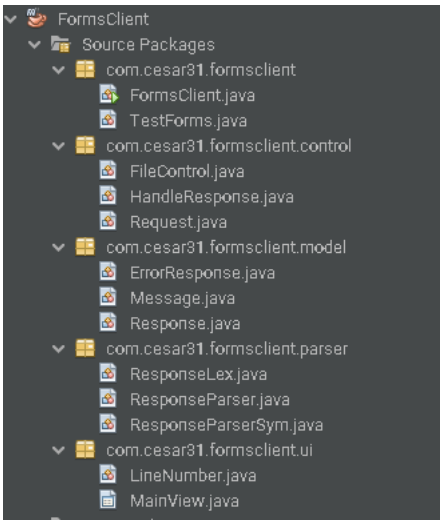
Parte esencial del servidor es enviar respuestas a la aplicación cliente según la resolución de las peticiones hechas por el mismo, dichas respuestas siguen una estructura similar a las peticiones que hace el cliente, por ejemplo:

```
<!ini_respuesta:"ERROR">
{
  "lexema" : null,
  "type" : "SINTACTICO",
  "line" : 11,
  "column" : 1,
  "description" : "Se encontro: null. Se esperaba: '>'."
}
<!fin_respuesta>
```

Puede existir el caso donde se encuentren más de una respuesta o solo una, según las peticiones hechas por el cliente y estructura correcta de estas así como de los datos correctos en las solicitudes. La aplicación cliente se encarga de realizar el análisis léxico y sintáctico de dichos mensajes y mostrarlos al usuario final de una manera más adecuada. La estructura de los componentes para analizar las respuestas se explican más adelante en la sección de la aplicación cliente.

Organización del Código Fuente de la aplicación cliente

La aplicación cliente que fue desarrollada también con Java(1.8.0_251), es una aplicación de escritorio desarrollada con la biblioteca gráfica Java Swing. La organización del código fuente viene distribuida de la siguiente manera:



El código fuente se divide en 5 paquetes el primero de ellos almacena el método main necesario para la ejecución del programa.

Posteriormente el package `com.cesar31.formsclient.control` tiene a las clases que estan a cargo de la lógica de la aplicación, por ejemplo métodos para guardar, abrir un nuevo archivo, escribir en un archivo y enviar las peticiones al servidor.

En el package `com.cesar31.formsclient.model` se guardan las clases POJO necesarias para el funcionamiento y conversión de los mensajes del servidor hacia estas mismas clases.

En `com.cesar31.formsclient.parser` estan almacenadas las clases necesarias para el análisis léxico y sintáctico de las respuestas del servidor.

Y por último, el package `com.cesar31.formsclient.ui` almacena las clases que tienen la parte visual de la aplicación cliente.

Análisis Léxico de las respuestas del servidor (Expresiones Regulares)

La estructura del archivo `response.flex`, donde se han establecido las instrucciones para el análisis léxico de las respuestas del servidor, siguen una estructura parecida a las que ya se explicaron.

```
/* Integer */
Integer = 0 | [1-9][0-9]*

/* Espacios */
LineTerminator = \r|\n|\r\n
WhiteSpace = {LineTerminator}|[\s\t\f]
```

Se tienen dos estados, YYINITIAL y STRING, este último para todos los tokens que inician y terminal con comillas. El estado YYINITIAL se expone a continuación.

```
<YYINITIAL> {

  "ini_respuesta"
  { return symbol(INI_R); }

  "fin_respuesta"
  { return symbol(FIN_R); }

  "ini_respuestas"
```

```
  { return symbol(INI_RS); }

  "fin_respuestas"
  { return symbol(FIN_RS); }

  "null"
  { return symbol(NULL, String.valueOf(yytext())); }

  {Integer}
  { return symbol(INTEGER, Integer.valueOf(yytext())); }

  "<"
  { return symbol(SMALLER); }

  "!"
  { return symbol(EXCL); }

  ">"
  { return symbol(GREATER); }

  "["
  { return symbol(LBRACKET); }

  "]"
  { return symbol(RBRACKET); }

  "{"
  { return symbol(LBRACE); }

  "}"
  { return symbol(RBRACE); }

  ":"
  { return symbol(COLON); }

  ","
  { return symbol(COMMA); }

  \"
  {
    string.setLength(0);
    yybegin(STRING);
  }

  {WhiteSpace}
  { /* Ignore */ }

}
```

En este estado se encuentran algunas palabras reservadas así como también algunos símbolos que se usan en la estructura de las respuestas del servidor.

Luego el estado STRING presentan la siguiente estructura.

```
<STRING> {

   \"
  {
    yybegin(YYINITIAL);
    return setType(STR, string.toString());
  }

    [^\n\r\"\\]+
  { string.append( yytext() ); }

  \\t
  { string.append('\t'); }

  \\n
  { string.append('\n'); }

  \\r
  { string.append('\r'); }

  \\\"
```

```
  { string.append('\"'); }

  \\
  { string.append('\\'); }

}
```

Para la clasificación de las palabras reservas entre comillas se utiliza el siguiente método.

```
private Symbol setType(int type, Object value) {
    String s = value.toString();

    switch(s) {
      case "request":
        return symbol(REQ, s);
      case "line":
        return symbol(LINE, s);
      case "column":
        return symbol(COLUMN, s);
      case "type":
        return symbol(TYPE, s);
      case "message":
        return symbol(MESSAGE, s);
      case "lexema":
        return symbol(LEXEMA, s);
      case "description":
        return symbol(DESC, s);

      default:
        return symbol(type, s);
    }
  }
```