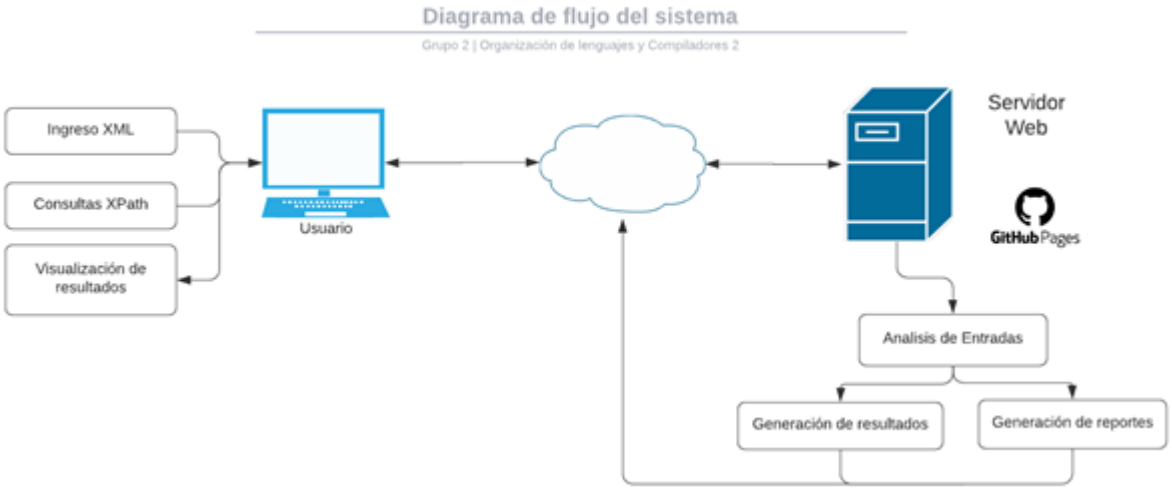
Manual Tecnico

Tytus es un software de licencia MIT traducción, consultas y ejecución de archivos de tipo XML, XPATH y XQuery

El diagrama de flujo del proyecto tiene la siguiente estructura



El usuario carga archivos de tipo XML, hace consultas usando el lenguaje de xpath y se le mostrarán los resultados, la página web y el analizador se encuentran alojados en un repositorio de github y se utiliza github pages para su despliegue, el analizador también generara reportes segun el archivo ejecutado

Tecnologías utilizadas para el proyecto

HTML 5 (HyperText Markup Language, versión 5) es la quinta revisión importante del lenguaje básico de la World Wide Web, HTML. HTML5 específica dos variantes de sintaxis para HTML: una «clásica», HTML (text/html), conocida como HTML5, y una variante XHTML conocida como sintaxis XHTML 5 que deberá servirse con sintaxis XML (application/xhtml+xml).12 Esta es la primera vez que HTML y XHTML se han desarrollado en paralelo. La versión definitiva de la quinta revisión del estándar se publicó en octubre de 2014

CSS es un lenguaje de diseño gráfico que permite definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Es muy usado para establecer el diseño visual de los documentos web e interfaces de usuario escritas en HTML

JavaScript (abreviado comúnmente JS) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos,2 basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas 3 y JavaScript del lado del servidor (Server-side JavaScript o SSJS). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, que esencialmente añade tipos estáticos y objetos basados en clases. Anders Hejlsberg, diseñador de C# y creador de Delphi y Turbo Pascal, ha trabajado en el desarrollo de TypeScript.1 TypeScript es usado para desarrollar aplicaciones JavaScript que se ejecutarán en el lado del cliente o del servidor, o extensiones para programas (Node.js y Deno).

TypeScript extiende la sintaxis de JavaScript, por tanto cualquier código JavaScript existente debería funcionar sin problemas. Está pensado para grandes proyectos, los cuales a través de un compilador de TypeScript se traducen a código JavaScript original.

TypeScript soporta ficheros de definición que contengan información sobre los tipos de librerías JavaScript existentes, similares a los ficheros de cabeceras de C/C++ que describen la estructura de ficheros de objetos existentes. Esto permite a otros programas usar los valores definidos en los ficheros como si fueran entidades TypeScript de tipado estático. Existen cabeceras para librerías populares como jQuery, MongoDB y D3.js, y los módulos básicos de Node.js.

Jison es, en escencia, un clone del generador grammatical Bison y Yacc, pero en javascript. Incluye su propio analizador lexico modelado en base a Flex (analizador lexico para JAVA). Fue creado originalmente por Zach Carter para ayudar el estudio de un curso de compiladores.

Browserify es una herramienta de JavaScript de código abierto que permite a los desarrolladores escribir módulos estilo Node.js que se compilan para su uso en el navegador

para la generacion de los archivos js se utilza el siguiente comando
npm run build, en donde build esta configurado en el json con el comando ts

El comando para unificar todos los js convertidos es el siguiente
browserify main.js --standalone load > bundle.js
esto nos ayuda a tener un solo archivos js en donde utilizaremos todas las funciones creadas

Configuración de archivos y gramática

Para generar los archivos .js ejecutar: tsc

Para generar el archivo .js de la gramática, ejecutar: jison .\Gramatica\gramatica.jison (Al ejecutar el comando se genera el archivo gramatica.js, mover/remplazar el archivo en la carpeta: dirt/Gramatica)

Para usar Browserify:
Instalar globalmente: npm install -g browserify Una vez ya hayamos generado los archivos .js con tsc, colocarnos en la carpeta dirt y ejecutar: browserify main.js --standalone load > bundle.js (bundle.js va a cargar todos los export, podemos acceder en html --> load.'nombre export'. Ejecutar cada vez que hagamos una modificación en los archivos typescript)

Los archivos que copiamos manualmente a la carpeta dist al ejecutar tsc no se sobrescriben o borran.

Validación de errores

Para cada gramática se trata de que el analizador al encontrar un error no termine su ejecución, para eso se agregan producciones adicionales con la palabra reservada **error,** sustituyendo a uno o más tokens.


```
    }
| error                {
    var xPathSyntaxAscError = new Error(
        yytext,
        this._$.first_line,
        this._$.first_column,
        'Error sintáctico'
    )
    xPathAscSyntaxErrors.push(xPathSyntaxAscError)
}
;
```

Uso de browserify

El uso de browserify es para vincular los archivos .js con los archivos .html, existen imports que no pueden ser generados por parte del cliente entonces browserify une todos los archivos para que estos puedan ser referenciados desde uno solo. Cada vez que se realiza un cambio se debe sustituir el bundle.js para que contenga los últimos cambios generados.

Funcionamiento de acciones en gramática

Para la mayoría de acciones se crea una clase y a la clase se le añaden valores de inicio, esta clase es la que se devuelve en cada producción, adicional para graficar el árbol, en cada producción se crea un objeto nodo.

```
        {
    $$ = new Element('', TypeElement.ALL, [], 1, @1.first_column)
    var nodo = {
        name: 'EL',
        val: 'EL',
        children: [{name: '*', val: '*', children: []}]
    }
    $$ = {...$$, Nodo: nodo}
}
| '..'              {
    $$ = new Element('', TypeElement.PARENT, [], 1, @1.first_column)
    var nodo = {
        name: 'EL',
        val: 'EL',
        children: [{name: '..', val: '..', children: []}]
    }
    $$ = {...$$, Nodo: nodo}
}
```

```
VALUES_CONT_ATRB :
    VALUE                                                   {
                                                                $$ = { nodo: new Nodo('VALUES_CONT_ATRB',[$1.nodo]), objeto: $1.objeto};
                                                                reporteGramatical.push('<tr> <td>VALUES_CONT_ATRB</td> <td>VALUE</td> </tr>');
                                                            }
    | lt                                                    {
                                                                $$ = { nodo: new Nodo('VALUES_CONT_ATRB',[new Nodo('lt',[])]), objeto: $1};
                                                                reporteGramatical.push('<tr> <td>VALUES_CONT_ATRB</td> <td>lt</td> </tr>');
                                                            }
    | apost                                                 {
                                                                $$ = { nodo: new Nodo('VALUES_CONT_ATRB',[new Nodo('apost',[])]), objeto: $1};
                                                                reporteGramatical.push('<tr> <td>VALUES_CONT_ATRB</td> <td>apost</td> </tr>');
                                                            }
;

/* CONTENIDO DE ATRIBUTO CON COMILLA SIMPLE */
```