Ingeniería Informática.

MWRegEx.

Monitor Web de Expresiones Regulares.

Daniel Ocaña Heredia.

Tutor: Dr. Zenón Hernández Figueroa.

Las Palmas de Gran Canaria.

Trabajo Fin de Grado de Ingeniería en Informática de la Universidad de Las Palmas de Gran Canaria presentado por el alumno:

Daniel Ocaña Heredia.

Título del Proyecto:
MWRegEx: Monitor Web de Expresiones Regulares.

Tutor:
Dr. Zenón Hernández Figueroa.

Fecha: 26/11/2014

Lugar: Administración de La Escuela de Ingeniería Informática de la Universidad de Las Palmas de Gran Canaria.

“Some people, when confronted with a problem, think, "I know, I'll use regular expressions." Now they have two problems”.

Jamie Zawinski.

A mi familia y amigos.

Agradecimientos.

Me gustaría agradecer en primer lugar a mi tutor Dr. Zenón Hernández Figueroa toda la confianza depositada en mí desde el comienzo de este Trabajo de Fin de Grado, tanto a nivel personal como académico. Gracias por su apoyo constante, a los profesores que me han formado y a los miembros de la Universidad de Las Palmas de Gran Canaria, este trabajo se ha convertido en un aprendizaje diario y en un magnifico reto.

El día a día en nuestra Universidad, las largas prácticas, las horas de estudio, los momentos más duros y también los más agradables, durante estos tres últimos años, no habrían sido lo mismo sin Dariel González, José Alberto Jinoria y Armando Pérez, que comenzaron siendo mis compañeros y se han convertido en grandes amigos. Por su paciencia, su ayuda y su amistad, muchas gracias.

Sin duda he podido llegar hasta aquí y soy como soy gracias a las dos personas que me han criado, cuidado, enseñado y querido siempre mi hermana Suzel Ocaña y mi madre Mirta Heredia. A ellas les debo todo y jamás podre agradecerles lo mucho que han hecho por mí cada día en todos los aspectos, ni expresar lo mucho que les quiero.

Casi al comenzar la Universidad conocí a la persona con la que comparto todas mis alegrías y mis penas, con la que disfruto día a día y a la que quiero, mi novia Yessica. Son muchas las experiencias vividas a lo largo de la carrera, gracias a su ayuda todo ha sido más fácil y agradable. Me gustaría poder alcanzar muchas más metas y conseguir nuevos retos a tu lado.

Antes de llegar aquí, tuve la oportunidad y la gran suerte de vivir una de las etapas más bonitas de mi vida junto a mis amigos de siempre. A pesar de la distancia siempre les querré. A Alexei, José Carlos y a todos y cada uno de los demás, gracias por lo que hemos vivido juntos y por quererme como soy.

Agradezco también al resto de mis familiares el apoyo y el cariño que he recibido a lo largo de mi vida. De cada uno he aprendido muchísimo en todos los sentidos y les querré toda la vida.

A familiares, amigos y profesores, muchas gracias.

Daniel.

Resumen del Trabajo de Fin de Grado en Español e Inglés.

El Trabajo de Fin de Grado, Monitor Web de Expresiones Regulares (*MWRegEx*), es una herramienta basada en tecnologías web, desarrollada usando el entorno Visual Studio. El objetivo principal de la aplicación es dar apoyo a la docencia de expresiones regulares, en el marco de la enseñanza del manejo de ristras de caracteres en las asignaturas de programación del Grado en Ingeniería Informática. La aplicación permite obtener el dibujo de un autómata de una expresión regular, facilitando su comprensión; además, permite aplicar la expresión a diferentes ristras de caracteres, mostrando las coincidencias encontradas, y ofrece una versión de la expresión adaptada a su uso en literales *string* de lenguajes como Java y otros. La herramienta se ha implementado en dos partes: un servicio web, escrito en *C#*, donde se realizan todos los análisis de las expresiones regulares y las ristras a contrastar; y un cliente web, implementado usando tecnología asp.net, con *JavaScript* y *JQuery*, que gestiona la interfaz de usuario y muestra los resultados. Esta separación permite que el servicio web pueda ser reutilizado con otras aplicaciones cliente.

El autómata que representa una expresión regular esta dibujado usando la librería *Raphaël JavaScript* que permite manejar los elementos *SVG*. Cada elemento de la expresión regular tiene un dibujo diferente y único para así diferenciarlo.

Toda la interfaz gráfica de usuario está internacionalizada de manera tal que pueda adaptarse a diferentes idiomas y regiones sin la necesidad de realizar cambios de ingeniería ni en el código. Tanto el servicio web como la parte cliente están estructurados para que se puedan agregar nuevas modificaciones sin que esto genere una onda expansiva a lo largo de las diversas clases existentes.

The final work, Regular Expression Monitor (*MWRegEx*), is a tool based on web technologies, developed using Visual Studio environment. The main objective of this application is to support the teaching of regular expressions, in the field of education in string management of characters in the subjects of programming Computer Science. The application allows for the drawing of an automaton of a regular expression, facilitating the understanding; also allows apply the expression to different character strings

showing the matches found and provides a version of the expression adapted to use in string literals and other languages such as Java. This tool has been implemented in two parts: a web service, written in *C#*, where all analyzes of regular expression and the strings are made to contrast; and a web client implemented using asp.net technology, *JavaScript* and *Jquery*, which manages the user interface and displays the results. This separated code allows the web service be reused with other client applications.

The automaton that represents a regular expression is drawn using *Raphael Javascript* library that can handle *SVG* elements. Each element of the regular expressions has a different and unique drawing to that way differentiate each one.

Graphic user interface is internationalized so that it can adapt to different languages and regions without needing engineering changes or code. Both the web service and the client side structure can add new features without generating a shock wave along the existing classes.

Índice de contenido

1.-Introducción.

Una expresión regular es una secuencia de caracteres que define un patrón de búsqueda en textos. Los lenguajes de programación modernos ofrecen instrumentos para la manipulación de ristras de caracteres usando expresiones regulares que constituyen una potente herramienta en el tratamiento de textos, al permitir configurar de forma fácil y flexible complejas operaciones de búsqueda y sustitución por patrones genéricos.

Para un programador es muy importante el dominio de estos instrumentos, para lo cual resulta imprescindible un adecuado conocimiento de las expresiones regulares que los sustentan. Por ello, las herramientas que puedan ayudar a adquirir ese conocimiento y a desarrollar las competencias asociadas al mismo resultan de particular interés en la docencia de programación. Numerosos problemas de tratamientos de ristras requieren la localización en un texto de subristras que cumplen un determinado patrón: fechas, números de DNI, números de la seguridad social, e-mails, direcciones web, etc.

Por este motivo, los profesores que han venido impartiendo la asignatura de Fundamentos de Programación del Grado en Ingeniería Informática ven la necesidad de desarrollar una aplicación en la cual los alumnos puedan obtener el autómata finito determinista de una expresión regular dada y a su vez una mejor compresión de las mismas. Además poder obtener los grupos de ocurrencia en un texto dado.

Como motivación personal me decidí por hacer este proyecto porque resultaba para mi muy importante ampliar mis habilidades en el desarrollo de aplicaciones web usando como Framework *ASP.NET*. Además de un mejor sentido de las expresiones regulares ya que hoy en día son un sistema cómodo, rápido y potente de realizar un filtrado sobre un determinado caso, y a su vez obtener grupos reducidos y específicos, excluyendo resultados que no coincidan con el patrón dado. En general las expresiones regulares suelen ser especialmente útiles para los programadores (*PHP, Perl, Java, .NET*) que pueden desarrollar y reconocer patrones complejos de una forma sencilla.

El proyecto propuesto pretende desarrollar una herramienta web con el propósito de apoyar el aprendizaje de las expresiones regulares que sea lo suficientemente abierta para poder adaptarse a las necesidades presentes y futuras de las asignaturas de programación la EII donde se imparten o usan expresiones regulares.

Teniendo en cuenta que el servicio web ofrecido no trabaja con datos de carácter personal, como podría ser en caso de direcciones IP, correo electrónico, nombre de personas etc. No infringe ninguna legislación vigente, así como tampoco la regulación del uso de las cookies en Internet.

1.1-Objetivos.

El objetivo fundamental del proyecto "MWRegEx", Monitor Web de Expresiones Regulares es desarrollar una aplicación avanzada basada en tecnologías web, que permita a los usuarios visualizar el autómata representado por una expresión regular y a su vez los resultados de cada coincidencia de una expresión regular en una ristra dada, además todas las capturas que coinciden con el grupo de captura, en orden empezando por el más interno de la izquierda, de manera ágil y sencilla a través de una única interfaz. Las tareas a abordar para realizar dicha aplicación son las siguientes:

1.-Crear un servicio web que utiliza *XML* en *ASP.NET*.

2.-Crear un Sitio Web independiente que utiliza este Servicio Web.

1.3-Aportaciones.

La aplicación a desarrollar permitirá tener separado el Servicio Web de la aplicación web, dando lugar a que otras aplicaciones puedan usar el servicio web cuando lo precisen, utilizando las ventajas que brinda el lenguaje de programación *C#* y su integración con *HTML 5.0*, se obtendrá una solución que sirva de apoyo para la enseñanza de Expresiones Regulares en las asignaturas de programación. Toda la interfaz gráfica de usuario será automáticamente traducida dependiendo del lenguaje detectado en la maquina huésped. Se pueden añadir idiomas tantos como sean necesarios así como modificar cualquier archivo de idioma, dado que los recursos son independientes y solo se carga la versión de lenguaje apropiado.

1.4-Estructura del documento.

Con el fin de facilitar la lectura del documento se presenta en este apartado su estructura general y aspectos más relevantes.

Tras haber realizado la introducción del Trabajo y haber destacado los principales objetivos del mismo, se desarrollan a continuación cada una de las fases. En primer

lugar se presentan los estudios realizados acerca del estado del arte y las herramientas existentes antes de comenzar el proyecto. Posteriormente, se detallan los recursos utilizados, tanto hardware como software, para desarrollar la aplicación y su posterior despliegue.

En el siguiente capítulo se detallara la metodología seguida a lo largo del ciclo de vida del mismo, así como la planificación y la temporización.

En el capítulo 5 y 6 se detallan el análisis, diseño y desarrollo de la aplicación, de forma exhaustiva, para finalizar el documento con los resultados y conclusiones obtenidas, junto con el posible trabajo futuro que podría realizarse. Además se indica la bibliografía consultada.

En cada una de las fases presentadas en este documento se podrán encontrar diferentes apartados, organizados de manera que facilite la lectura y compresión de la misma. A lo largo de la memoria se hará referencia a la bibliografía consultada utilizando el código correspondiente a cada libro o sitio Web entre paréntesis por ejemplo: (MRE 3rd Edition) hace referencia al libro escrito por Jeffrey E.F. Friedl sobre “Mastering Regular Expressions, 3rd Edition Understand Your Data and Be More Productive”.

1.5-Justificación de las competencias específicas cubiertas.

Con el desarrollo de este Trabajo de Fin de Grado de la carrera Grado en Ingeniería Informática, se deben cubrir las competencias asignadas a éste, las cuales son: CII01, CII02, CII04, CII018 y TFG01. A continuación, se listan cada una de ellas junto con una explicación de cómo se han cubierto.

1. CII01: Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.

   Para la realización de este trabajo fue necesario diseñar e implementar una aplicación web. De igual manera, fue necesario enfrentarse a la necesidad de tomar decisiones a la hora de la elección de un framework o tecnología a usar. Finalmente, se desarrolló una aplicación que cumple con la legislación y normativas vigentes para el desarrollo del software.

2. CII02: capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social.

   Para llevar a cabo esta aplicación se realizó una correcta planificación y dirección del proyecto. El mismo consistió en el desarrollo de una aplicación web que se desplego en un entorno real. Además se capturaron limitación y posibles mejoras para el futuro cercano, para aumentar la usabilidad a medida que se desarrollen funcionabilidades nuevas.
3. CII04: capacidad para elaborar el pliego de condiciones técnicas de una instalación informática que cumpla los estándares y normativas vigentes.

   En el desarrollo de este documento se incluyeron en el capítulo tres los requisitos de hardware y software necesarios para el correcto desarrollo de esta aplicación y para su despliegue. Además, se especificó el marco legal en que se ampara el presente proyecto.
4. CII18: conocimiento de la normativa y la regulación de la informática en los ámbitos nacional, europeo e internacional.

   Para el desarrollo de este o cualquier proyecto, los desarrolladores deben estar al tanto de lo legal y conocer la normativa y la regulación de la informática en el ámbito nacional, de la Unión Europea, e internacional.

   TFG01: ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas.

   Luego de haber terminado el desarrollo del Trabajo de Fin d Grado, se debe hacer una presentación y defensa del mismo, ante el tutor y un tribunal constituidos por miembros del claustro de profesores, para cumplir con lo establecido con el TFG01. De esta manera se demuestra la consistencia en el ámbito de las tecnologías específicas de la Ingeniería Informática, entre las cuales se destacan la captura de requerimientos, análisis y diseño del software, así como la implementación, pruebas, manejo de base de datos, administración de servidores y diseños de interfaces de usuarios. Todos estos conocimientos han sido adquiridos durante la formación que se obtuvo a lo largo de la carrera Grado en Ingeniería Informática.

2.-Estado del Arte.

En este apartado se presenta un estudio de la situación actual. Del estado de los sitios web más relevantes que de una forma u otra permiten permite representar o ensayar las expresiones regulares. Se ha observado una gran variedad de formatos y estructuras, la mayoría se organizan en las mismas secciones y apartados, contienen interfaces sencillas que permiten escribir una expresión regular y luego escribir un texto para finalmente buscar en dicho texto los patrones de búsqueda que dicta la expresión regular; se dictan a continuación las herramientas que se usan en esos portales web son *Phyton, Ruby on Rails, JavaScript, HTML* etc. Del mismo modo se ha encontrado un sitio web *(www.regexper.com)* que permite hacer un dibujo de la expresión regular en forma de autómata, pero que no permite en caso de que la expresión regular contenga un error realizar un dibujo. También existe un sitio web *(https://www.debuggex.com/)* en cual además de dibujar el autómata para una expresión regular, es capaz de reconocer en un texto las cadenas que cumplen con el patrón de la expresión regular, y reconoce en la expresión regular dibujando el carácter en rojo los errores que pueda tener dicha expresión regular aunque solo es capaz de reconocer un error a la vez .En todos estos portales web carecen de la función de multilenguaje y que a su vez permitan obtener el autómata finito de la expresión regular y también poder buscar ocurrencias en textos usando esa expresión regular.

2.1- Sitios Web.

En este apartado se analizaran en sentido general las aplicaciones encontradas que permiten obtener el autómata finito determinista de una expresión regular o encontrar las coincidencias de las mismas en un texto. Teniendo en cuenta aspectos relevantes como por ejemplo si están desarrolladas como aplicaciones web o aplicaciones finales. Ver cuales dibujan el autómata finito y cuáles no lo hacen. Para el caso de aquellas aplicaciones que dibujen el autómata ver como realizan el dibujo de la misma en el caso de que la expresión regular tenga un error. Comprobar cuales realizan búsquedas de ocurrencias y si se identifican claramente los grupos. Distinguir cuáles de las aplicaciones son multilenguaje o mono lenguaje.

2.1.1- Regexper.

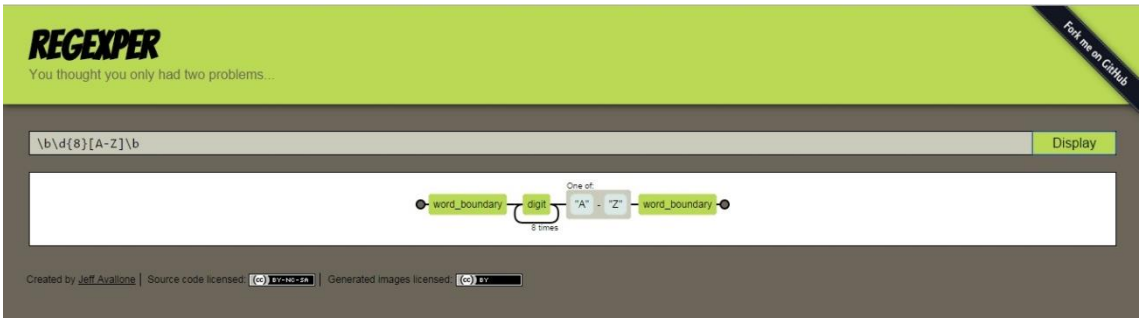


Ilustración 1 Aplicación de Regexper.

Esta aplicación Web permite realizar el dibujo de un autómata finito, dado una expresión regular, es una aplicación muy potente, a continuación mostramos los aspectos positivos y las limitaciones:

Positivo:

1. Permite obtener el autómata finito de una expresión regular.

Limitaciones:

1. No permite realizar búsquedas de ocurrencias usando la expresión regular.
2. No es una aplicación multilenguaje.
3. Los errores no son representados gráficamente.
4. Es una aplicación final.

2.1.2- Regexr.

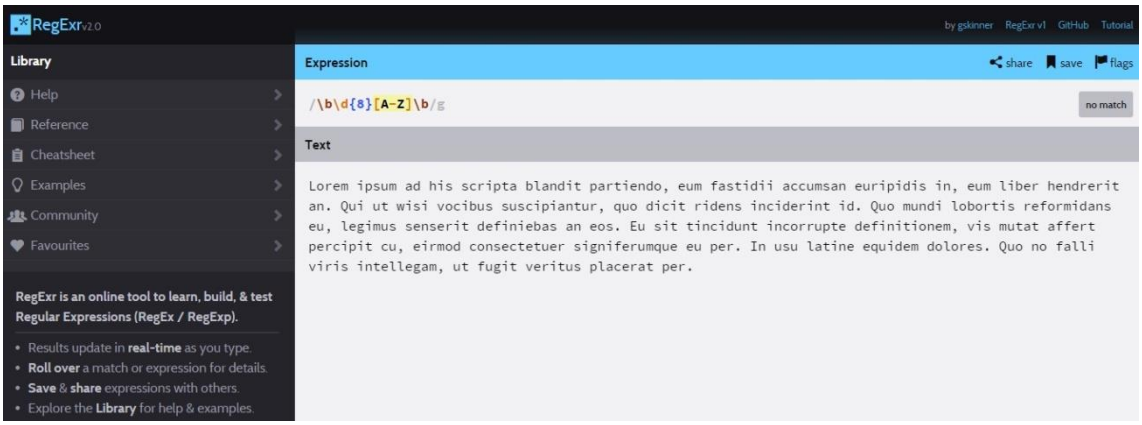


Ilustración 2 RegExr es una herramienta en línea para aprender, construir y probar expresiones regulares.

Positivo:

1. Resalta las ocurrencias de una expresión regular en un texto.
2. Con el evento onmouseover, da una pequeña explicación de lo que significa cada símbolo en la expresión regular.

Limitaciones:

1. No permite obtener el autómata de la expresión regular para una mejor comprensión de la misma.
2. No es una aplicación multilenguaje.
3. Es una aplicación final.

2.1.3- RegexPlanet.

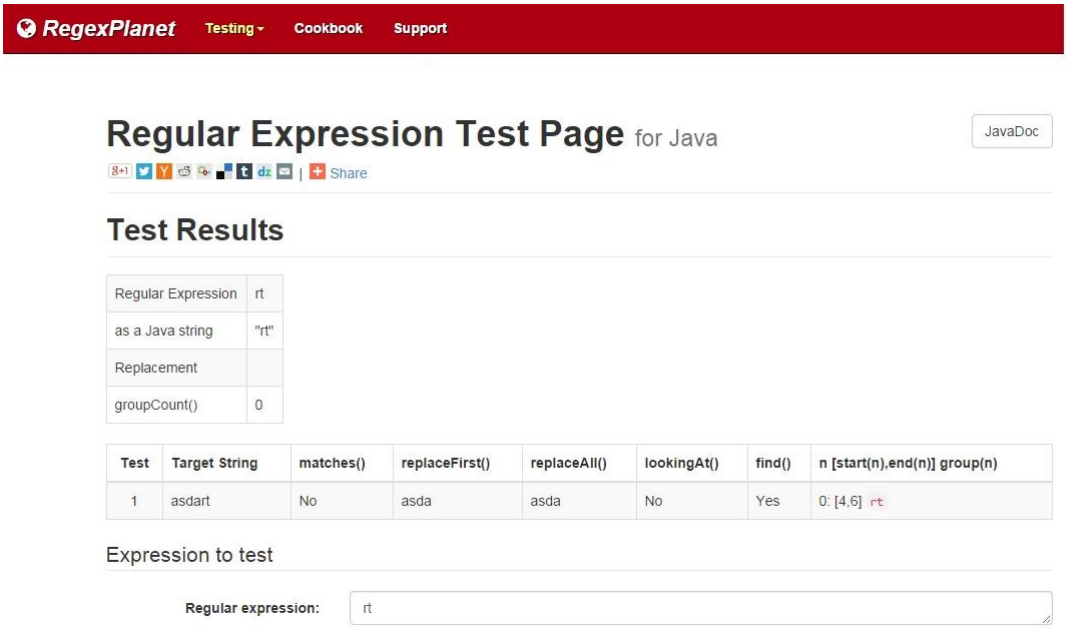


Ilustración 3 RegexPlanet.

Positivo:

1. Permite probar tus expresiones regulares en varios lenguajes entre ellos (*PHP*, *Perl*, *Java*, *JavaScript*, entre otros lenguajes).
2. Permite hacer búsquedas y reemplazar cadenas dentro de cadenas.

Limitaciones:

1. No permite obtener un dibujo para una mejor comprensión de la propia expresión regular.
2. No es una aplicación multilenguaje.
3. Es una aplicación final.

2.1.4- DebbugexBeta.

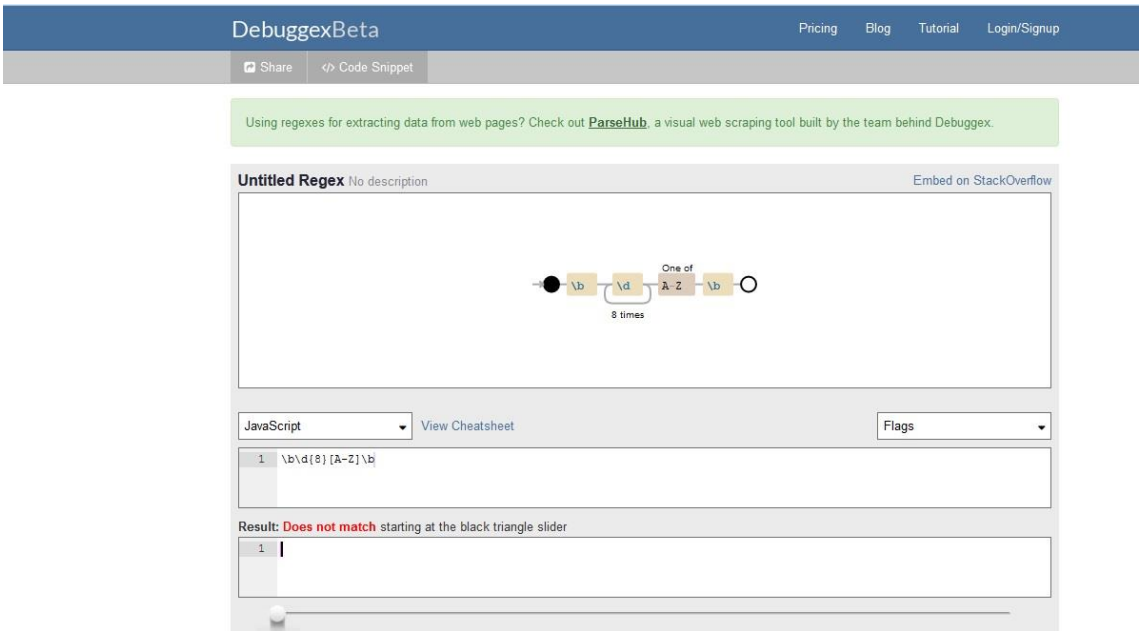


Ilustración 4 DebbugerBeta.

Positivo:

1. Dibuja el autómata finito de una expresión regular.
2. Permite buscar ocurrencia de una expresión regular en un texto.

Limitaciones:

1. No es una aplicación multilenguaje.
2. No está realizada como servicio web.
3. Los errores no son tratados como un dibujo y solo se reconoce un error a la vez.

Después del análisis de cada una de las aplicaciones anteriores, se puede concluir que para el caso de aquellas que realizan un dibujo, solo una aplicación permite obtener los grupos de las ocurrencias *(2.1.4- DebbugexBeta.)*, además en caso de que la expresión regular contenga un error no realizan dibujo alguno. Por otra parte las que realizan solo búsquedas de patrones en textos, además de no dibujar el autómata tampoco realizan búsquedas de grupos. Y finalmente todas las aplicaciones analizadas son mono lenguaje y son aplicaciones finales.

3.-Estudio de Herramientas.

En este capítulo se describen los recursos utilizados para el desarrollo de la aplicación y los recursos mínimos para el correcto funcionamiento de la misma.

3.1- Recursos Software.

3.1.1- Herramientas.

1.-*Visual Studio* es un entorno de desarrollo integrado para sistemas operativos *Windows*, que soporta múltiples lenguajes de programación (por ejemplo *C++*, *C#*, *.NET*, etc.). Permite a los desarrolladores crear aplicaciones, sitios web, así como servicios web en cualquier entorno que soporte la plataforma *.NET*. Por lo que se puede crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos, consolas, etc. [9].

Visual Studio ha sido el entorno principal de desarrollo del proyecto. Dada la posibilidad de crear una aplicación potente y de alto rendimiento en muy poco tiempo.

2.- *JQuery* es una biblioteca *JavaScript* rápida, pequeña y rica en funciones. Contribuye a la manipulación del documento *HTML*, control de eventos, animación y *Ajax* mucho más simple, con una *API* muy fácil de usar que funciona perfectamente en muchos navegadores. Con una combinación de flexibilidad y extensibilidad.
JQuery se ha usado en el proyecto para que el desarrollo de la aplicación en el lado del cliente se vea enriquecido y que a su vez sea compatible con varios navegadores. Además permite simplificar la manera de interactuar con los documentos *HTML*, manipular el árbol *DOM*, manejar eventos y para desarrollar animaciones a páginas web. [8].

3.- *IIS Express* es un servidor web que ofrece una infraestructura de gran fiabilidad, capacidad de manejo y escalabilidad para aplicaciones Web para el sistema operativo Microsoft Windows. [56]
Este servicio convierte a una PC en un servidor web para Internet o una Intranet, por lo que fue usado en el proyecto para realizar pruebas de funcionalidad y depuración durante el desarrollo del mismo.

4.- Raphaël JavaScript Library, el uso esta librería simplifica el trabajo con gráficos vectoriales en el web. Utiliza objetos *SVG*, cada uno de ello es también un objeto del *DOM*, lo que permite conectar eventos *JavaScript* a estos objetos. [5].

La librería Raphaël es usada para crear el gráfico vectorial en la web, que muestra el autómata finito de la expresión regular.

5.-*Internet Explorer* (IE), navegador web desarrollado por Microsoft, también es una interfaz de usuario *FTP*. Fue el navegador utilizado para realizar las pruebas y la depuración dado su magnífica integración con el entorno de desarrollo (*Visual Studio*), ya que permitía la depuración del código *JavaScript* en tiempo real. También se ha realizado pruebas de funcionamiento de la aplicación sobre otros navegadores (*Mozilla Firefox 33.0.3, Google Chrome 38.0.2125.111, Safari 5.1.7*).

6.-*ASP.NET* es un *Framework* para aplicaciones web desarrollado y comercializado por Microsoft. Permite construir sitios web dinámicos, aplicaciones web y servicios web *XML*. *ASP.NET* está construido sobre el *Common Language Runtine*, permitiendo a los programadores escribir código asp.net usando cualquier lenguaje admitido por el *.NET* Framework. *ASP.NET* fue el modelo de desarrollo Web utilizado en el proyecto, para generar y representar de forma dinámica las páginas web.

3.2 - Lenguajes de Programación.

1.- *JavaScript* (JS) es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. El estándar de *JavaScript* es *ECMAScript*. A partir de 2012, todos los navegadores soportan completamente *ECMAScript* 5.1.Los navegadores más antiguos soportan al menos *ECMAScript* 3. Una sexta revisión del estándar está en proceso. [7].

Mediante *JavaScript* se han desarrollado funcionalidades avanzadas para la representación de los valores obtenidos del servidor.

2.- *HTML 5* Es el lenguaje de marcado de la *World Wide Web*. Fue diseñado principalmente para describir semánticamente documentos científicos, sin embargo su diseño general y adaptaciones en los últimos años han permitido que sea utilizado para representar otros tipos de documentos. Permite múltiples *JavaScripts* ejecutándose en

paralelo en la misma página, hace posible la inserción de audio y video de forma directa y la geolocalización de usuario.

Este lenguaje será el utilizado para presentar los contenidos y herramientas al usuario en cada uno de los apartados que componen el sistema.

3.- *CSS 3* Hojas de estilo en cascada (*Cascading Style Sheets*), es un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla y la forma en que se va a imprimir. Esta forma de descripción, ofrece a los desarrolladores el control total sobre el estilo y formato de sus documentos. *CSS* se utiliza para dar estilo a documentos *HTML* y *XML*, separando el contenido de la presentación. Permite a los desarrolladores controlar el estilo y el formato de múltiples páginas web simultáneamente. Cualquier cambio de estilo marcado para un elemento en el *CSS*, afectara a todas las páginas vinculadas a ese *CSS* en las que aparezca ese elemento.

De esta manera, se ha utilizado el *CSS* para dar formato a cada uno de los documentos *HTML* desarrollados a través de sus etiquetas. Mediante esta tecnología, se facilita la presentación de contenidos y se ofrece una interfaz de usuario más accesible y clara.

4.- C# Es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma *.NET*. Está diseñado para compilar diversas aplicaciones que se ejecutan en *.NET* Framework. Es un lenguaje simple, eficaz y con seguridad de tipos. Las numerosas innovaciones de *C#* permiten desarrollar aplicaciones rápidamente y mantener la expresividad y elegancia.

En este proyecto se ha usado para programar los servicios. Se han programado todas las clases que modelan la aplicación utilizando este lenguaje de programación, ya que *C#* permite generar programas sobre la plataforma *.NET*.

5.-*XML* busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien identificadas, y que esas partes se componen a su vez de otras partes. Una etiqueta *XML* consiste en una marca hecha en el documento, que señala una porción de éste como un elemento. Un pedazo de información con un sentido claro y definido. Las etiquetas tienen la forma <nombre>, donde nombre es el nombre del elemento que se está señalando.

Esta tecnología fue usada para describir los datos que se intercambian entre el servidor y el cliente, con ello se facilita la realización de declaraciones de contenido y la obtención de resultados en varias plataformas. Además los archivos de recursos están basado en *XML* se utilizan para almacenar cadenas de la interfaz de usuario que deben ser traducidas a otros idiomas.

3.3 – Recursos Hardware.

El proyecto fue desarrollado usando un ordenar personal marca Acer con las siguientes características:

- Procesador Intel Core i3.
- Memoria RAM 4 GB DDR2.
- Pantalla de 15,4 pulgadas.
- Sistema Operativo: Windows 8 profesional.

3.4 – Hardware del Servidor.

Para poder alojar el servicio ofrecido por el sistema, será necesario contar con un servidor web. Este equipo deberá poseer las características hardware necesarias para proveer un servicio accesible por múltiples usuarios al mismo tiempo.

Al tratarse de un servicio basado en Web, será primordial el tiempo de respuesta de la maquina a la hora de responder a las múltiples peticiones por parte de los usuarios del sistema. El servidor indicado deberá poseer al menos las siguientes características:

- Procesador: Pentium 4 o superior.
- RAM: 2 GB o superior.
- Disco Duro: 5 GB disponible mínimo.
- Red LAN que soporte TCP/IP (en general, Internet).

Los requisitos del cliente son muy sencillos simplemente con un ordenador o dispositivo móvil con conexión a internet y navegador web, ya se puede conectar al sitio web.

4.-Metodología.

La metodología empleada en el desarrollo software del trabajo ha sido el modelo en espiral, propuesto originalmente por Boehm en 1988, es un modelo de proceso de software evolutivo que conjuga la naturaleza iterativa de construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial.

Proporciona el potencial para el desarrollo rápido de versiones incrementales del software. En el modelo espiral, el software se desarrolla en una serie de versiones incrementales. Durante las primeras iteraciones, la versión incremental podría ser un modelo en papel o prototipo. Durante las últimas iteraciones, se producen versiones cada vez más completas del sistema diseñado.

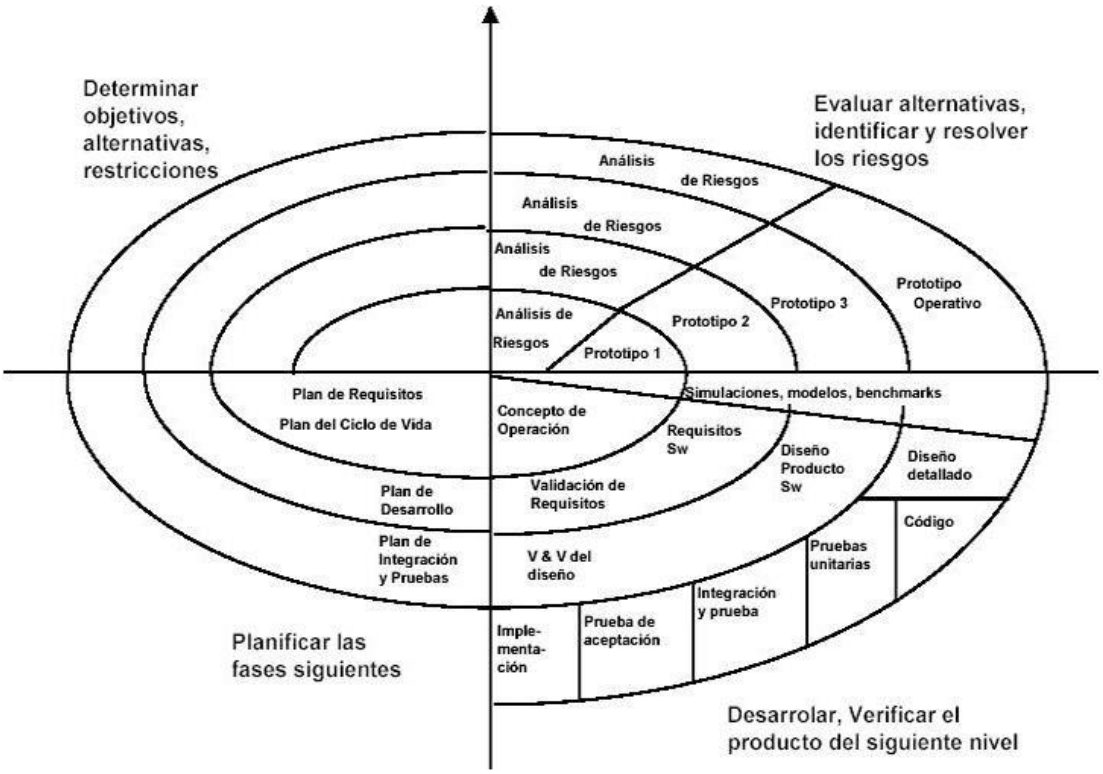


Ilustración 5 Fases del modelo en espiral.

El modelo en espiral se divide en un número de actividades de marco de trabajo, también llamadas regiones de tareas. Generalmente, existen entre tres y seis regiones de tareas. El modelo en espiral contiene seis regiones de tareas:

Comunicación con el cliente: se trata de las tareas requeridas para establecer comunicación entre el desarrollador y el cliente.

Planificación: las tareas requeridas para definir recursos, el tiempo y otra información relacionadas con el proyecto.

Análisis de riesgos: las tareas requeridas para evaluar riesgos técnicos y de gestión.

Ingeniería: las tareas requeridas para construir una o más representaciones de la aplicación.

Construcción y acción: las tareas requeridas para construir, probar, instalar y proporcionar soporte al usuario, como documentación y práctica.

Evaluación del cliente: las tareas requeridas para obtener la reacción del cliente según la evaluación de las representaciones del software creadas durante la etapa de ingeniería e implementada durante la etapa de instalación.

Cuando empieza este proceso evolutivo, se gira alrededor de la espiral en la dirección de las agujas del reloj, comenzando por el centro. El primer circuito de la espiral puede producir el desarrollo de una especificación de productos; los pasos siguientes en la espiral se podrían utilizar para desarrollar un prototipo y progresivamente produce ajustes en el plan del proyecto. El coste y la planificación se ajustan con la realimentación ante la evaluación del cliente.

El modelo en espiral es un enfoque realista del desarrollo de sistemas de software de sistemas a gran escala. Como el software evoluciona, a medida que progresa el proceso, el desarrollador y el cliente comprenden y reaccionan mejor ante riesgos en cada uno de los niveles evolutivos. El modelo en espiral utiliza la construcción de prototipos como mecanismo de reducción de riesgos, pero lo que es más importante permite a quien lo desarrolla aplicar el enfoque de construcción de prototipos en cualquier etapa de evolución del producto. Mantiene el enfoque sistemático de los pasos sugeridos por el ciclo de vida clásico, pero lo incorpora al marco de trabajo iterativo que refleja de forma más realista el mundo real.

4.1-Lenguaje de Modelado.

El lenguaje de modelado utilizado a lo largo del proyecto será *UML* o Lenguaje Unificado de Modelado. Permite expresar un modelo de análisis utilizando una notación de modelado con unas reglas sintácticas, semánticas y prácticas.

Es un lenguaje grafico para visualizar, especificar, construir y documentar un sistema. *UML* ofrece un estándar para describir un modelo del sistema, incluyendo aspectos conceptuales tales como procesos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. Se emplea para definir un sistema, para detallar los artefactos en el mismo, para documentar y construir.

UML cuenta con varios diagramas, de los cuales muestran diferentes aspectos de las entidades representadas.

4.2 – Planificación y temporización.

En este apartado se detallan las horas empleadas para la realización del trabajo.

Fases	Horas
INICIO	**40H**
Comunicación con el cliente	20H
Planificación	5H
Análisis de riesgos.	5H
Ingeniería.	5H
Construcción y acción.	5H
Evaluación del cliente.	0H
ELABORACIÓN	**80H**
Comunicación con el cliente	20H
Planificación	10H
Análisis de riesgos.	10H
Ingeniería.	10H
Construcción y acción.	25H
Evaluación del cliente.	5H
CONSTRUCCIÓN	**150H**

Comunicación con el cliente	20H
Planificación	10H
Análisis de riesgos.	15H
Ingeniería.	15H
Construcción y acción.	80H
Evaluación del cliente.	10H
TRANSICIÓN	**30H**
Comunicación con el cliente	5H
Planificación	5H
Análisis de riesgos.	5H
Ingeniería.	5H
Construcción y acción.	10H
Evaluación del cliente.	0H
TOTAL	**300H**

Tabla 1: Planificación por fases y flujos.

4.3- Presupuesto.

A continuación se detalla el presupuesto para efectuar el proyecto. Se destacan los elementos necesarios para la realización del mismo, así como el coste estimado para cada uno de ellos.

Coste del *Hardware*: Es el coste correspondiente a los equipos utilizados.

Coste del *Software*: Gastos derivados de la utilización de las herramientas empleadas para la realización del proyecto.

4.3.1- Coste del Hardware.

Para este proyecto se tienen en cuenta los siguientes elementos de hardware para su desarrollo.

Elemento	Coste
Ordenador Portátil Acer	400€
Servicio de Internet ADSL	160€
Servidores hosting	Gratuito

Total Hardware	560

Tabla 2: Desglose de los costes de hardware.

4.3.2- Coste del Software.

El coste de los siguientes elementos es el correspondiente a la licencia de los mismos.

Elemento	**Coste**
Windows 8 Professional	160€
Microsoft Office 2007	130€
Visual Studio	Licencia MSDN del Departamento de Informática y Sistemas (ULPGC).
Total Hardware	290

Tabla 3: Desglose de los costes de software.

4.3.3- Presupuesto total del proyecto.

Elemento	**Coste**
Coste del Hardware	560
Coste del Software	290
Total	850

Tabla 4: Desglose de los costes.

5.-Análisis.

5.1.-Preámbulo.

Las expresiones regulares son secuencias de caracteres que forman un patrón, que podemos comparar con otra secuencia de caracteres para encontrar coincidencias o realizar operaciones de sustitución. Cada expresión regular tiene un autómata finito asociado.

Las expresiones regulares están disponibles en casi cualquier lenguaje de programación, pero aunque su sintaxis es relativamente uniforme, cada lenguaje usa su propio dialecto.

Las expresiones regulares utilizan un motor de búsqueda, estos motores se clasifican en dos grandes grupos: Motores para el programador y Motores para el usuario.

Los programas que permiten realizar búsquedas sobre algún contenido, son diseñados para que el usuario, usando expresiones regulares, pueda realizar búsquedas sin embargo esto hace que los usuarios tengan que saber cómo representar correctamente la expresión regular adecuada para poder obtener todo el beneficio del programa. Los ejemplos más comunes son *grep* (programa de los sistemas operativos *Unix*.); *PowerGrep* que es la versión de grep para los sistemas operativos *Windows*.

Los motores que son utilizados por los programadores, permiten realizar el proceso de búsqueda de modo que sea posible utilizarlo muchas veces para un propósito específico. Algunos ejemplos de estos motores son: *Java*: existen varias bibliotecas hechas para *java* de modo que permiten el uso de las expresiones regulares. Del mismo modo *JavaScript* a partir de la versión 1.2 da soporte a las Expresiones Regulares. Y el *Framework* utilizado en este proyecto *.NET* también provee un conjunto de clases mediante las cuales es posible utilizar las expresiones regulares para hacer búsquedas, reemplazar cadenas y validar patrones.

Al utilizar las expresiones regulares como motor de búsqueda esto permite especificar todas las opciones; como encontrar palabras que comiencen por unas letras especificas o que en el medio de una cadena se encuentre un digito y muchas otras combinaciones, como un lenguaje que permite enviarle al motor de búsqueda exactamente lo que deseamos buscar en todos los casos, sin necesidad de tener opciones adicionales al realizar la búsqueda.

Para especificar las distintas opciones a buscar se utiliza un lenguaje o convención por el cual se le transmite al motor de búsqueda el resultado exacto esperado. Al tener este lenguaje se le dan un significado especial a una serie de caracteres; por lo que estos caracteres no serán buscados literalmente en el texto sino que se buscara lo que estos caracteres signifiquen. Estos caracteres son conocidos algunas veces como “meta caracteres”; los mismos, y como el motor de búsqueda los interpreta, se detallan en el siguiente apartado.

5.2.-Descripción de las Expresiones Regulares.

En este apartado se describirán los diferentes elementos que se usan para crear una expresión regular.

El punto “.”; se interpreta como cualquier carácter, o sea busca cualquier carácter sin incluir los saltos de línea; aunque existe la opción en *.NET* para especificar que busque todos los caracteres incluidos saltos de líneas.

El punto se utiliza de la siguiente forma: Si se le dice al motor de búsqueda “c.s” en la cadena “la casa donde se casaron” el motor de búsqueda encontrara “casa”, “casa”ron. Nótese que el punto representa un solo carácter. En caso de buscar más caracteres será necesario utilizar las repeticiones.

La barra inversa ‘\’ denotara que el siguiente carácter de la expresión adquiera un significado especial o que deje de tenerlo. Por lo que la barra inversa nunca se utiliza por sí sola, sino en combinación con otros caracteres. Al utilizarlo por ejemplo con el punto este deja de tener su significado y se comporta como un literal. De la misma forma sucede si se coloca esta barra seguida de cualquiera de los caracteres especiales; estos dejan de tener su significado y se convierten en caracteres de búsqueda literal. Por ejemplo si se usa “\t” esto representa un tabulador. [2].

Los corchetes “[]”; la función de los corchetes es representar “clases de caracteres” o lo que es lo mismo agrupar caracteres en conjuntos. Dentro de los corchetes se usa el guion “-” para indicar rangos de caracteres, adicionalmente los meta caracteres pierden su significado y se convierten en literales cuando se encuentran dentro de los corchetes. El único carácter que es necesario anteceder con la barra inversa dentro de los corchetes es la propia barra inversa. Los corchetes permiten buscar palabras aún si están mal escritas. No importa la cantidad de caracteres que escribas dentro de los corchetes, el motor de búsqueda ira un carácter a la vez.

Por ejemplo, la expresión regular “ambici[oó]n” permite encontrar en cualquier texto la palabra “ambición” aunque se haya escrito con o sin tilde.

La barra "|" sirve para indicar una o varias opciones. Es utilizada comúnmente en conjunto con otros caracteres especiales.

La expresión regular "verde|azul|blanco" permitirá encontrar cualquiera de los nombres de esos colores que se encuentra en la expresión regular.

El símbolo "^" obliga a empezar la búsqueda por el termino, lo que representa el inicio de cadena, de la misma forma el símbolo $ representa e final de cadena. Cuando se utiliza en conjunto con los corchetes permite encontrar cualquier carácter que NO se encuentre dentro del grupo indicado.

En primer lugar su funcionalidad como carácter individual representa el inicio de línea por tanto si se utiliza la expresión regular "^a" el motor de búsqueda encontrara todas las palabras que den inicio con la letra minúscula "a". Cuando se utiliza en conjunto con los corchetes de la siguiente forma "[^\w]" permite encontrar cualquier carácter que NO se encuentre dentro del grupo indicado; por ejemplo cualquier carácter que no sea alfanumérico o un espacio.

Los paréntesis nos permiten agrupar caracteres, de forma parecida a como lo hacen los corchetes, sin embargo, mientras los corchetes sirven para crear conjuntos de caracteres, los paréntesis crean subexpresiones, lo que introduce una dimensión recursiva en el análisis de las expresiones que aumenta su complejidad.

- Los caracteres especiales conservan su significado dentro de los paréntesis.
- Los grupos establecidos con paréntesis establecen un punto de referencia para el motor de búsqueda,
- Utilizados en conjunto con la barra permite hacer búsquedas opcionales. Por ejemplo la expresión regular "(día|tarde|noche)" permite buscar textos que den indicaciones por medio de las tres opciones, mientras que si solo escribimos "día|tarde|noche" se encontraría "día" en la palabra "ardía", no pudiendo cumplir con este propósito.
- Utilizados en conjunto con otros caracteres especiales, ofrecen otras funcionalidades.

En el siguiente ejemplo "(p|m)adre" se agrupan los símbolos en un solo patrón; para de esta forma poder encontrar tanto la palabra madre como padre. En expresiones más

complejas los paréntesis nos ayudan a memorizar los patrones encontrados, por ejemplo "(11+)" guarda la primera aparición de al menos dos unos seguidos.

El signo de interrogación "?" tiene varias funciones dependiendo donde se coloque dentro de la expresión regular. La primera de ellas es especificar que una parte de la búsqueda es opcional. La otra opción es usarlo en conjunto con el carácter "!", al principio de unos paréntesis, para buscar elementos que no están seguidos de aquellos que especificamos en la expresión regular.

Por ejemplo, la expresión regular "visu?al" permite encontrar tanto "visual" como "visal".

Las llaves "{}", generalmente las llaves son usadas para determinar la cantidad de elementos que tiene que encontrar el motor de búsqueda del carácter que precede a las llaves por ejemplo la expresión regular de la siguiente forma \s{2}, el motor buscara un espacio en blanco dos veces.

En cambio sí lo ponemos de la siguiente forma \s{2,4} indica al motor de búsqueda que como máximo debe aparecer 4 veces la expresión regular "\s" y como mínimo dos.

Para aquellos casos en los que algunas veces no se conoce con claridad la cantidad de veces que se repite lo que se busca su utilizan los siguientes metacaracteres.

El asterisco "*" sirve para encontrar algo que se encuentra repetido entre cero o más veces. El comportamiento por defecto del asterisco es encontrar la mayor cantidad posible de caracteres que se correspondan con el patrón de búsqueda.

Si hacemos la búsqueda "a*u", los resultados que podemos encontrar serían "aula", "aaula" o incluso "uno" o "cuello", porque el asterisco implica que el carácter no tiene por qué aparecer.

El signo de suma "+" su utiliza para encontrar una cadena que se encuentre repetida una o más veces a diferencia del asterisco la cadena a buscar tiene que estar al menos una vez.

Por ejemplo si buscamos "a+u", los resultados podrían ser "aula" o "auula" pero no ni "uno" ni "cuello".

6.-Desarrollo del Trabajo.

En este apartado se presentaran cada uno de los aspectos de interés de la fase de desarrollo. El código fuente al completo del desarrollo llevado a cabo se adjunta a esta memoria en un CD.

6.1-Sitio Web.

La implementación del sitio web ha sido realizada utilizando *Web Forms*, el cual forma parte de los cuatro modelos de programación que se puede utilizar en *ASP.NET* para crear aplicaciones Web. Los otros modelos son *ASP.NET MVC*, páginas web ASP.NET, y paginas simples para aplicaciones en *ASP.NET*.

Web Forms son páginas que los usuarios solicitan a través del navegador, estas páginas se pueden escribir usando una combinación con *HTML*. Cuando un usuario hace una petición de una de estas páginas, se compila y se ejecuta en el servidor por el *Framework*, y luego el mismo *Framework* genera el código *HTML* que el navegador puede reproducir. Una página *Web Forms* en *ASP.NET* presenta información al usuario en cualquier navegador o dispositivo cliente.

El entorno de *Visual Studio* permite separar los controles del servidor para de esta forma diseñar correctamente el *Web Forms*. Por lo que resulta más fácil configurar las propiedades, métodos y eventos para los controles de la página. Estos métodos y propiedades van definir el comportamiento de la página. En este proyecto para escribir el código del servidor que maneja la lógica de la página, se ha utilizado el lenguaje *.NET C#.*

Que son los *Web Forms*:

1. Basado en tecnología de *Microsoft ASP.NET*, en el que el código que se ejecuta en el servidor genera dinámicamente una página Web que el navegador presenta en cualquier dispositivo cliente.
2. Compatible con cualquier navegador o dispositivo móvil. La página que se genera es lista para ser compatible para funciones tales como estilo, diseño etc.

3. Compatible con cualquier lenguaje de programación soportado por *.NET Commom Language Runt*inme como por ejemplo *Visual Basic y Microsoft Visual C#.*
4. Construido en el propio *Framework* lo cual proporciona todos los beneficios del *Framework*, incluyendo un entorno administrativo, seguridad en los tipos y herencia. Además es flexible porque se pueden agregar controles creados por terceros usuarios.

Que ofrecen los *Web Forms*:

1. Separa los diferentes lenguajes de programación utilizados en la interfaz de usuario de los de la lógica de la aplicación.
2. Conjunto de controles de servidor para tareas comunes, incluyendo acceso a datos.
3. Soporte para comandos del lado del cliente que se ejecutan en el navegador.
4. Soporte para variedad de otras capacidades, como por ejemplo enrutamiento, seguridad, rendimiento, internacionalización, prueba, depuración y tratamiento de errores.

El sitio web se ha realizado utilizando *Web Forms* dado que daba unas ventajas sobre otros modelos de crear una aplicación web, a continuación menciono las siguientes ventajas:

Es compatible con el modelo de eventos que conserva el estado a través de *HTTP*, además me ofrecía decenas de controles en la parte del servidor.

Utiliza el patrón *Controller* que añade funcionabilidad a páginas individuales. Y además también usa vistas que puede hacer la gestión de la información de estado mucho más fácil.

Funciona bien para pequeños equipos de desarrollo web que necesitan tomar ventaja de la gran cantidad de componentes disponibles para el desarrollo rápido de aplicaciones.

En general es menos complejo y requiere menos codificación que el modelo *MVC*. Al utilizar esta tecnología se obtiene como resultado una aplicación robusta y escalable.

6.2- Interfaz.

La interfaz la componen todos los elementos de la pantalla que permiten al usuario realizar acciones sobre el Sitio Web. Por lo mismo, se considera parte de la interfaz a sus elementos de identificación, de navegación, de contenidos y de acción.

Todos estos elementos están preparados para ofrecer servicios al usuario, con el fin de que éste obtenga lo que desea. Por lo anterior, cada uno de los elementos que están integrados dentro de la interfaz está pensados para causar un efecto sobre el usuario. Teniendo en cuenta esto, durante la explicación de este apartado se podrá ver la ubicación relativa de todos los componentes de la aplicación web.

Ilustración 6 Interfaz inicial.

La Ilustración 6 muestra tal cual la interfaz inicial que el usuario observa cuando entra en el sitio web. En este punto solo se encuentran dos controles un input inicial que está destinado para que el usuario escriba la expresión regular y un botón para interactuar con la página y a su vez visualizar la expresión regular escrita. Además encontraremos el título de portal web así como los nombres de los desarrolladores y el logo de la Universidad de Las Palmas de Gran Canaria y de la Escuela de Ingeniería Informática.

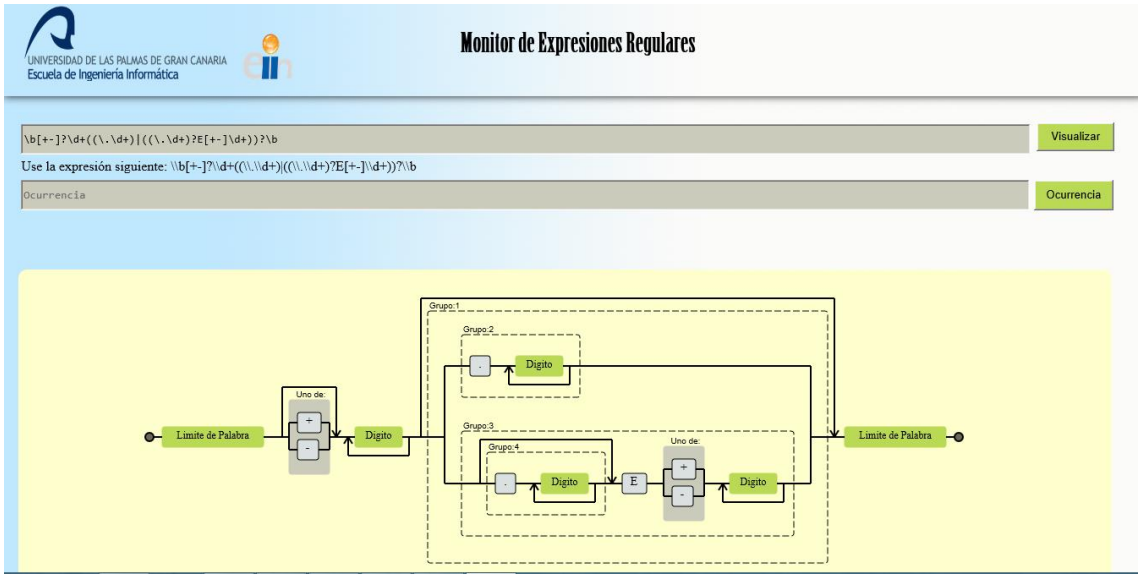


Ilustración 7 Expresión regular para un número real o entero.

Una vez que el usuario escribe una expresión regular y da *click* en *Visualizar*, para el caso en el que la expresión regular este bien escrita, lo siguiente que se puede visualizar es que la expresión regular se le han doblado las barras para que dicha expresión regular se pueda trasladar al lenguaje de programación *Java*. Además de esto se crea un nuevo input en el cual el usuario podrá escribir un texto para buscar las ocurrencias de la expresión regular en dicho texto, esto se podrá hacer dando click en el botón *Ocurrencia* que también acaba de aparecer. Finalmente se muestra el área que va a contener el lienzo del dibujo, así como el autómata asociado a la expresión regular escrita inicialmente.

El dibujo estará formado por rectángulos que contienen cada grupo, carácter o metacarácter. Estos rectángulos se podrán diferenciar por su color de fondo por ejemplo los metacaracteres tendrá un color verde de fondo, el grupo delimitado por paréntesis tendrá un rectángulo con líneas discontinuas que forme el grupo. Los caracteres simples tendrán de fondo un color claro y los corchetes serán de color gris.

Estos rectángulos están conectados por líneas que indican el camino a seguir por el motor de búsqueda. Y para cada elemento que tenga repeticiones se colocaran en la parte inferior derecha de su respectivo rectángulo.

El inicio y el final del autómata esta delimitados por dos círculos de color oscuro son lo que delimitan respectivamente inicio y final del autómata.

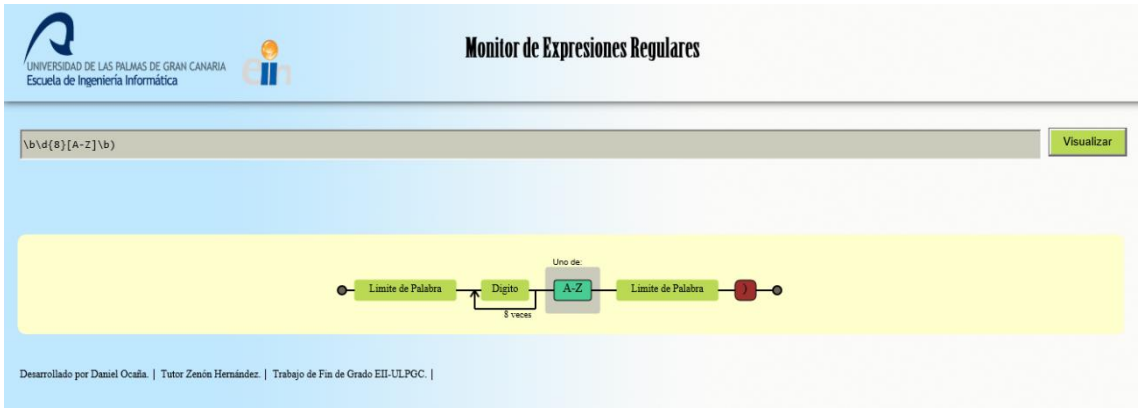


Ilustración 8 Error en la expresión regular.

En la Ilustración 8 se puede observar como en caso de que la expresión regular contenga algún error no se muestra ningún input para buscar ocurrencias dado que no se pueden buscar cuando la expresión regular está mal escrita. Además de esto, el dibujo muestra un rectángulo parpadeando entre color rojo y blanco donde se localiza el error encontrado para que de esta forma el usuario sepa en qué parte de la expresión regular se encuentra el error. Tampoco se agrega en este caso la expresión regular con las barras dobladas.

Volviendo al caso en el que la expresión regular esta correctamente escrita, en la Ilustración 9 podemos comprobar cómo se localizan las ocurrencias, dada una expresión regular y un texto que se introduce en el segundo *input*. Vemos como se genera una tabla con todas las ocurrencias en el texto de dicha expresión regular así como los grupos que la componen.

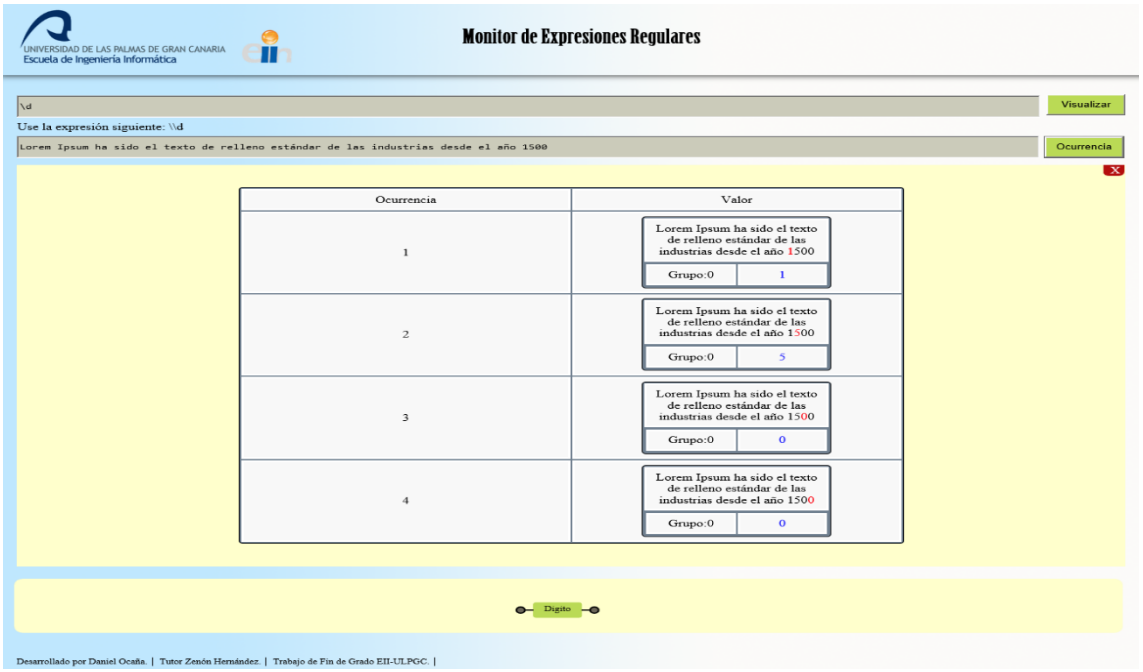


Ilustración 9 Tabla de Ocurrencias.

Todos estos elementos que conforman la interfaz se usan en conjunto con un filtro para mostrar solamente ciertos bloques cuando el usuario modifique la expresión regular que se encuentra en el primer input, de esta forma si el usuario modifica la expresión regular que se acaba de visualizar eso conlleva a que el dibujo no sea el correcto por lo tanto esos bloques no me muestran hasta que el usuario vuelva a dar click en *Visualizar* y entonces si se muestre el autómata correspondiente a esa nueva expresión regular. Si una vez modificada la expresión regular, el usuario deshace esos cambios y regresa a la expresión regular original, pues entonces se volvería a mostrar el autómata correspondiente a esa expresión regular. Este filtro es de gran ayuda al usuario ya que siempre se mostrara el dibujo correspondiente a la expresión regular que contenga el primer input.

6.3-Estructura y diseño.

Los *Web Forms* están diseñados por dos componentes: la parte visual (el archivo *ASPX*), y el código detrás del Forms que reside en un archivo de clase independiente.

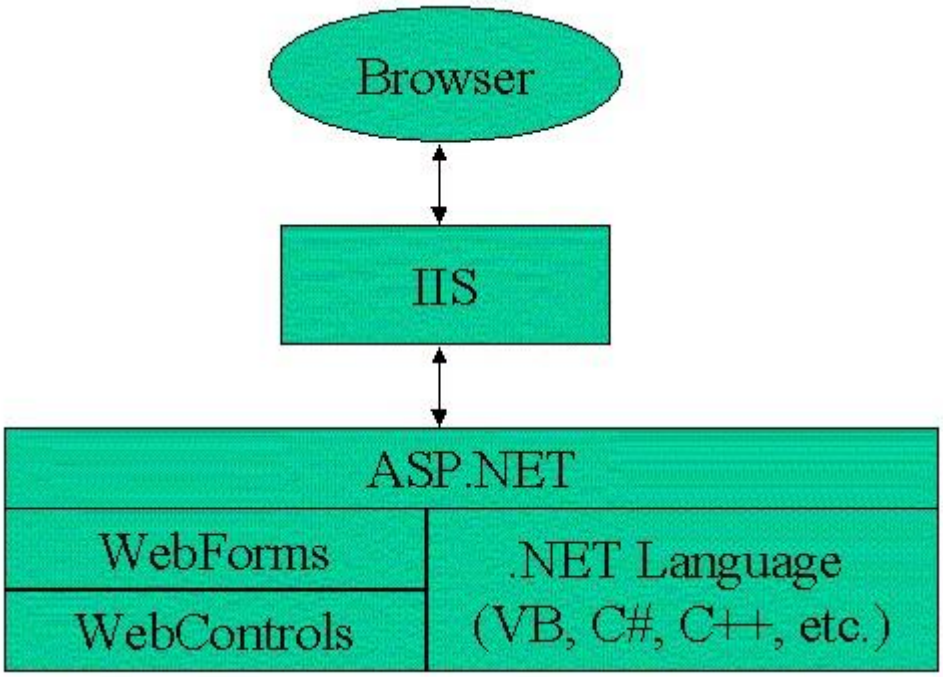


Ilustración 10 Web Forms forman parte de ASP.NET.

En la parte visual, se colocan los controles en el *Web Forms* para crear la interfaz de usuario. Los controles que se usan determinan que propiedades, eventos y métodos se pueden utilizar, hay dos tipos de controles que se pueden usar para crear la interfaz: los controles *HTML* y los propios controles que tienen los *Web Forms*.

6.3.1- Implementación de la funcionalidad.

Para este proyecto se ha utilizado los controles tipo *HTML*. En la interfaz de la aplicación se puede ver el primer control. El primer control es de tipo *TextBox* ya que este control es usado para crear un área donde el usuario pueda escribir la expresión regular que desea monitorizar.

Ilustración 11 Control de tipo TextBox.

Una vez que el usuario escribe una expresión regular y da click en *Visualizar*, se captura mediante el evento *click* usando *Jquery*, y lo primero que se realizar es restaurar todas las variables globales a su valor inicial ya que anteriormente podían tener otros valores que en este punto no interesan, además también se limpian todos las etiquetas de tipo div que se encuentran en la página.

Como se mencionó anteriormente los servicios web se componen de dos partes, el punto de entrada al servicio web y el código asociado a la funcionalidad del servicio. En *ASP.NET*, el archivo *.asmx* es un archivo que contiene una directiva de procesamiento *WebService* que hace referencia a las clases usadas, esta clase es pública y contiene dos métodos públicos marcados con el atributo *WebMethod*.

Siguiendo con los pasos realizados después de limpiar toda la página se llama al método de la clase *Service1*, *analizar*, el cual recibe como parámetro el texto escrito en el primer input, la clase Service1 instancia un atributo de tipo *Analizador* (esta clase programada en *C#* se detallaran en el siguiente Tema) y haciendo uso de la clase llama al método *analize* que es quien, a groso modo, se encarga de analizar la expresión regular escrita y retornar un elemento de tipo *RegexItem* que luego en la parte del cliente se analizara para su posterior dibujo.

Toda esta comunicación es posible dado que en el formulario de la página se define la ubicación del servicio Web mediante la declaración agregando un elemento <*asp: ServiceReference*> al elemento <*Services*> dentro del elemento <*asp:ScriptManager*> y a continuación estableciendo su atributo *Path*, como se muestra en el ejemplo siguiente.

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
    <Services>
        <asp:ServiceReference Path="~/Service1.asmx" />
    </Services>
</asp:ScriptManager>
```

Ilustración 12 Referencia al Web Service.

Una vez se analiza la expresión regular en el servidor, se llama a la función *success*, esta función se encarga de ir llamando a una serie de funciones que se encargaran de ir analizando la respuesta del servidor, primero se llama al método *calcularCoordenadas*, el cual obtiene como parámetro el resultado del servidor (response), este método utiliza la recursividad para de esta forma añadirle a cada RegexItem cuatro propiedades a través de *JavaScript*, estas propiedades son; la posición que va tener en el lienzo de dibujo expresado mediante los valores de las coordenadas en los ejes X e Y; el ancho del rectángulo que va a mostrar el *RegExItem*, que depende de su valor; y un alto que ocupara en el lienzo. Por ultimo este método se encarga de que en caso que el dibujo sea muy grande colocar un *scroll* al lienzo donde estará el dibujo del autómata.

Para calcular el tamaño de cada rectángulo en el lienzo antes de que se muestre, se llama a la función tam_*rectangulo* que recibe tres parámetros el texto que tiene el rectángulo adentro, el tamaño de la letra y el tipo de letra. Con estos datos creo un *SVG* invisible asignándole el texto pasado con el tamaño y el tipo de letra, entonces se puede obtener el ancho que ocupa ese texto en el lienzo invisible y de esta forma se sabe qué tamaño ocupa en el dibujo.

Luego de que cada elemento tiene sus coordenadas se crea el área sobre la cual se va a dibujar dado que ya conocemos el tamaño total del dibujo. Esta área se crea utilizando la librería *Javascript Raphael*.

Ya teniendo el lienzo se llama a la función que es la encargada de dibujar ,donde se plasmaran cada uno de los elementos del autómata, la función dibujar utiliza los atributos de cada *RegexItem* para identificarlos y así realizar un dibujo único para cada tipo de *RegexItem* haciendo uso de la coordenadas que se calcularon en la función anterior.

Luego de esto se realizan las líneas que conectan cada uno de los *RegexItem* usando la función *dibujarPath* y luego se le dibujan a aquellos elementos que tengan algún tipo de repetición las líneas correspondientes a su repetición, usando la función dibujar_Path_*Repeticiones* y finalmente se dibuja el circulo inicial y final del autómata finito, cabe destacar que todas estas funciones que dibujan utilizan el mismo lienzo y response.

Justo en el momento en el que se está dibujando cada elemento del response y antes de poner el Valor de cada carácter de la expresión regular, se llama a la función comprobar, esta función es la encargada de traducir al idioma seleccionado todos los textos que se escribirán en el dibujo SVG; la cual comprueba si existe el texto pasado por parámetro. Si es así, devuelve ese texto en el idioma correcto. En caso contrario, devuelve lo mismo que se le pasa por parámetro.

Una vez obtenido el dibujo correctamente, se ejecuta una función que es quien se encarga de en caso de que la expresión no contenga ningún error añadir otro control también de tipo *TextBox*, en el cual es donde introducirá un texto para buscar las ocurrencias de dicha expresión regular en un texto.

Ilustración 13 Input para buscar Ocurrencias.

En este segundo control, cuando el usuario escribe un texto y da *click* en *Ocurrencia*, se llama al método de la clase Service1 *analizar_ocurrencia*, el cual recibe como parámetro la expresión regular escrita en el primer *input* y el texto introducido en el segundo *input* y envía estos datos al método *analize_macht* de la clase *Analizador* este método busca todas las ocurrencias de dicha expresión regular en el texto y devuelve un *ArrayList* con todos los grupos de ocurrencia, así como la posición en la cadena original donde se encontró el primer carácter de la subcadena capturada, además la longitud de la subcadena capturada y la propia subcadena.

Luego de que el servidor devuelva estos datos se llama a la función *successMatch*, esta función se encarga de añadir un nuevo div a la página que contiene una tabla en la cual para cada ocurrencia una fila, mostrando todo el texto en negro y en rojo la ocurrencia. Y debajo se crea otra tabla con las ocurrencias de grupo, en este caso se muestran en color azul.

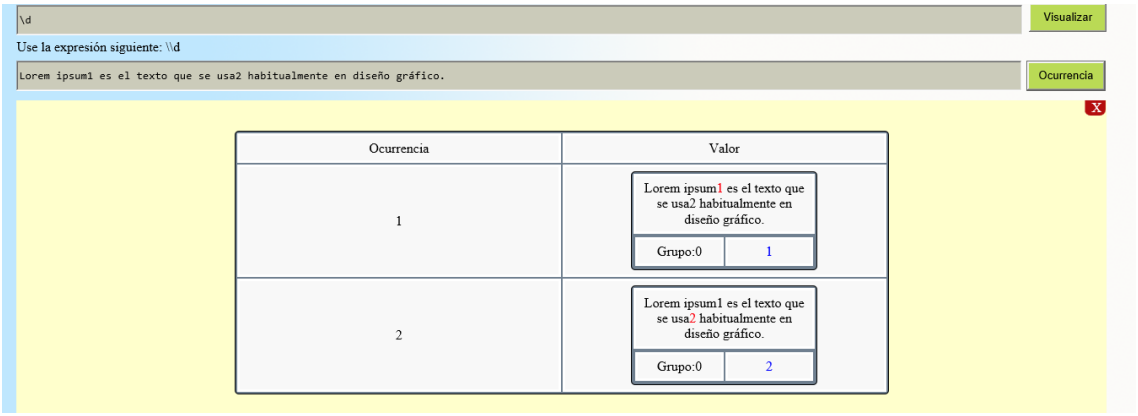


Ilustración 14 Tabla con las ocurrencias.

6.3.2- Internacionalización

Lo primero es preparar una solución personalizada [4]; las paginas *ASP.NET* permiten establecer dos valores de referencia para la internacionalización, correspondientes a *CULTURE* y *UICULTURE*. El valor *CULTURE* determina los resultados de las funciones que dependen de la referencia cultural, como el formato de la fecha, moneda, entre otras. El valor *UICULTURE* determina qué recursos se cargaran para la página.

Dado que para esta aplicación no interesa el valor que tenga *CULTURE*, solo nos centraremos en el valor de *UICULTURE*, y para obtener este valor se agrega a la cabeza de la página el valor de *UICULTURE* como auto de esta forma podemos obtener que lenguaje está utilizando el usuario en su navegador y mostrar así la interfaz adecuada. Un enfoque para dar solución a mostrar páginas en diferentes idiomas es crear una página para cada idioma, sin embargo esto puede ser mano de obra intensiva, y propenso a errores, y difícil de mantener a medida que cambia la página.

ASP.NET permite crear páginas en la cuales los contenidos y otros datos se mostraran a base de elección explicita o del lenguaje del usuario. Esto se conoce como archivos de recursos, estos archivos pueden ser locales o globales cuando se tiene varias páginas web. Y en tiempo de ejecución se obtienen los valores de los elementos a mostrar y se sustituye por los recursos del archivo correspondiente.

Un archivo de recurso es un archivo *XML* que contiene las cadenas que se desean traducir a diferentes idiomas o rutas de acceso a imágenes. Este archivo está formado por una tripla (Nombre, Valor, Comentario), donde cada tripla es un recurso individual. Por ejemplo para este proyecto tenemos el Nombre b1 y el Valor es *Visualizar* para el caso en español y *Display* para inglés, y su respectivo comentario.

Se crea un archivo de recurso independiente para cada idioma. Cada archivo tiene las mismas triplas (Nombre, Valor, Comentario). Estos archivos tienen una extensión *.resx* y se compila en tiempo de ejecución.

Para crear los archivos de recursos, se empieza por la creación de un archivo *.resx* base; y para cada idioma crear un nuevo archivo que tiene el mismo nombre que la pagina pero que incluya el idioma; por ejemplo:

- *regexper.aspx.resx*

  Archivo base. Este el archivo por defecto.

- *regexper.aspx.es.resx*

  Archivo de recurso para España.

- *regexper.aspx.en.resx*

  Archivo de recurso para Estados Unidos.

- *regexper.aspx.de.resx*

  Archivo de recurso para Alemania.

Finalmente en tiempo de ejecución *ASP.NET* usa el archivo adecuado para el valor obtenido por *UICULTURE*, por ejemplo si la cultura es español, pues se utiliza *regexper.aspx.es.resx*. Si no hay ninguna coincidencia se utiliza el archivo base.

Estos archivos de recursos locales se encontraran en la carpeta *App_LocalResources* que es un nombre reservado. Además del uso de *UICULTURE* para mostrar la interfaz adecuada dependiendo del idioma del navegador también se usan scripts para que se muestre todo el contenido que se crea dinámico en la página en el mismo idioma que la interfaz.

Dado que la página web será internacionalizada, debemos realizar la búsqueda del idioma del navegador a través de *Javascript*, de varias formas ya que no sabemos desde que navegador se hará la petición, es decir usando las propiedades *navigator.language* la cual funciona con todos los navegadores menos con el Internet Explorer, o *navigator.userLanguage* que será para el caso de *Internet Explorer*. Una vez que se conozca el idioma, la función *include* se encarga de añadirle al *head* de la página un script dependiendo del idioma, para no cargar al navegador con todos los textos posibles en todos los idiomas disponibles; en el cual encontraremos una dupla con clave y valor separados por dos puntos. Al realizarlo de esta manera se pueden añadir más idiomas simplemente copiando uno de los archivos de idiomas ya creados y cambiándole el valor por ejemplo; ya que se tiene el idioma español en el archivo *es.js*, en caso de añadir el idioma alemán seria copiar este archivo y cambiarlo a: *de.js* y traduciendo para cada clave el valor en el propio idioma.

6.3.3- Análisis de las clases.

En este apartado se detallaran las clases creadas para el servidor así como el uso en el mismo.

La clase *RegexItem* será quien represente a cada carácter de la expresión regular, para componer de esta forma una estructura que luego en el navegador cliente sea más fácil analizar. Esta clase contiene los siguientes parámetros, Tipo en el cual se pondrán los siguientes; cero en caso de ser una secuencia y no un carácter único, uno representara al elemento simple, dos serán los grupos formados por paréntesis, tres representara el

grupo alternativa o sea la barra, cuatro representaran a los corchetes, de esta forma tenemos los cuatro grandes grupos en los que cada carácter será de un único Tipo. Para aquellos caracteres que sean del mismo Tipo pero que necesitemos diferenciarlos, la clase contendrá un parámetro llamado Subtipo. El atributo Valor representara el carácter en sí mismo y las repeticiones serán las que tenga cada carácter. Además de esto la clase contendrá un *ArrayList* Componentes, pensado para aquellas partes de la expresión regular que componen un grupo. Fue necesario que esta clase heredara de la interfaz *ICloneable* para crear una copia profunda dado que en algunos casos era necesaria.

Esta clase será la que retorne el servidor con todos los datos necesarios para cada carácter de la expresión regular.

Teniendo en cuenta que se utilizaron las pruebas unitarias también fue necesario que esta clase modificara la implementación del método *Equals* para que se ajustara a las necesidades para poder comparar que dos elementos de tipo RegexItem sean iguales.

La clase My*_Match* será quien represente las búsquedas de patrones de una expresión regular en un texto, para ello contiene los siguientes atributos Valor que será el propio valor de una cadena encontrada; el índice será la posición donde comience la subcadena en la cadena original. La longitud de la cadena. Y contendrá un *ArrayList* que representaran los grupos encontrados.

La clase *Analizador*, es la clase principal del servidor contiene todos los métodos para ir diferenciando entre todos los caracteres de la expresión regular. El método principal de la clase es *analize*, ya que es quien recibe la expresión regular y la va separando por cada carácter y sus respectivos grupos. Además esta clase contiene métodos que se encargan de parametrizar las repeticiones de cada elemento dándole un sentido común a las mismas. También esta clase está capacitada para reconocer todos los posibles errores que pueda contener la expresión regular y capturarlos correctamente para que sea más fácil su representación en el lado del cliente.

6.4-Diagrama de clases.

Problema del contexto. ¿Cómo realizar una mejor estructura para una aplicación web donde se pueda conseguir la reutilización y la flexibilidad y evitando la duplicación de código?

Solución:

Utilizar el patrón *Controlador* para aceptar la entrada de la solicitud de la página, y de esta manera invocar el método solicitado en el modelo y determinar una vista correcta resultante. De esta forma se separa la lógica de cualquier código asociado con la vista. Y para este caso se crea una clase base común para todos los controladores de la página y de esta forma se evita el duplicar código y se aumenta la coherencia y capacidad de prueba. La ilustración siguiente muestra como el controlador de la página se refiere al modelo y a la vista.

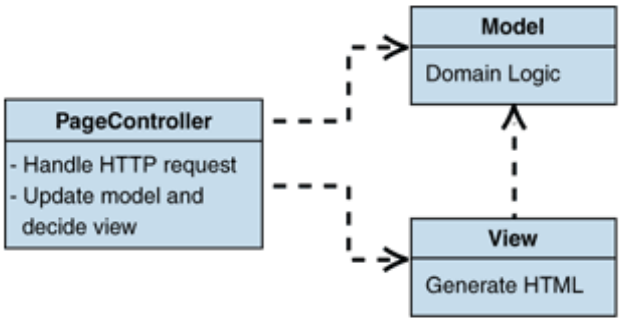


Ilustración 15 Estructura del patrón Controlador.

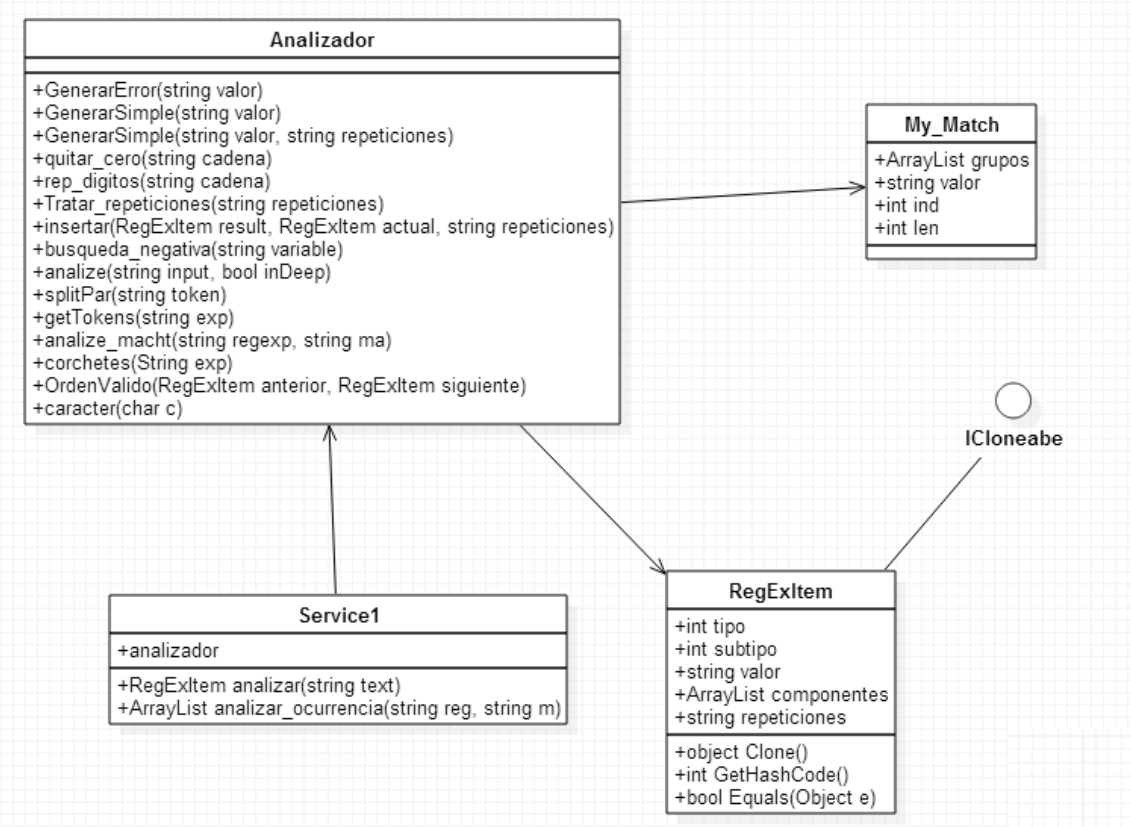
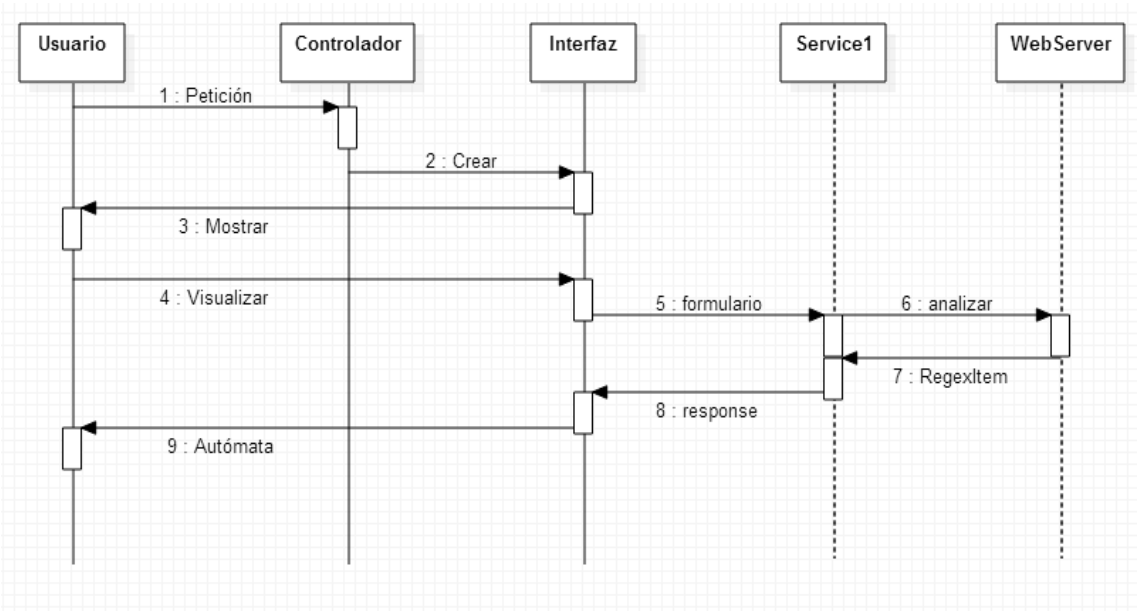


Ilustración 16 Diagrama de clases.

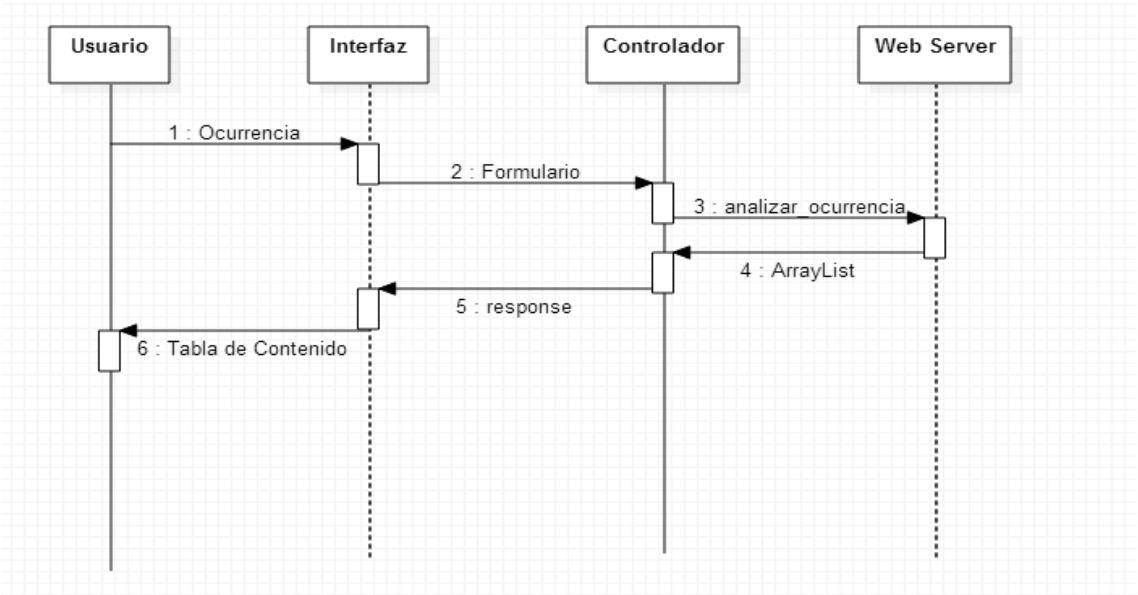
Con el diagrama anterior se observa como en el Servidor se tratara a las expresiones regulares de tal forma que lo que retorna el servidor sea un elemento de tipo *RegexItem* donde se contiene en cada posición un elemento de tipo *RegexItem* que tiene un Tipo, Subtipo, Valor, Repeticiones y que a su vez puede contener otro *ArrayList* dentro de Componentes.

6.5-Diagrama de Secuencia.

El siguiente es el Diagrama de secuencia para el flujo normal de datos para el caso *Visualizar*.



A continuación el Diagrama de secuencia para el flujo normal de datos para el caso de buscar Ocurrencias sobre un Texto.



6.6- Pruebas.

Es necesario realizar pruebas para con los resultados obtenidos de las mismas poder verificar si el sistema cumple o no con los requerimientos planteados en el inicio del desarrollo. Las pruebas realizadas se basan en el funcionamiento del Servicio Web. De forma que se pueda introducir una expresión regular y que se obtenga el dibujo correcto correspondiente a la expresión regular escrita y a su vez se puedan obtener en una tablas las ocurrencias de dicha expresión en un texto dado y a su vez los grupos de ocurrencias dentro de ese texto.

En este apartado se han utilizado las prueba unitarias [6] dado que es una forma de comprobar el correcto funcionamiento de los métodos implementados. Esto sirve para asegurarnos que cada uno de los módulos por separado funciona correctamente. La idea es escribir casos de prueba para cada función, de forma que cada caso sea independiente del resto.

Las características que deben cumplir estas pruebas para que tenga la suficiente calidad son: automatizable, esto significa que no debería requerirse intervención manual. Las pruebas tienen que ser completas para así cubrir la mayor cantidad de código posible. Nunca se deben crear pruebas que solo puedan ser ejecutadas una sola vez y además la ejecución de una prueba no debe afectar a la ejecución de otro prueba. Estas pruebas serán consideradas igual que el código, con la misma profesionalidad y documentación.

Al usar estas pruebas unitarias el objetivo es aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el trozo de código debe satisfacer. Al realizar estas pruebas aisladas proporcionan cinco ventajas básicas:

Fomentan el cambio: Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura o lo que es lo mismo **Refactorizar**.

Simplifica la integración: Puesto que permiten llegar a la fase de integración con un alto grado de seguridad en el código.

Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.

Los errores están más acotados y son más fáciles de localizar: dado que tenemos pruebas unitarias que pueden ver exactamente donde está el error.

En este procedimiento, se escribirán métodos de prueba unitaria para comprobar el comportamiento del método *analize*. El método, siempre tiene que devolver una estructura correcta para cada expresión regular en concreto.

Teniendo todo esto en mente la programación de pruebas unitarias utilizando *Visual Studio* y *C#* es muy sencilla, primero debemos crear un nuevo proyecto de tipo Prueba Unitaria.

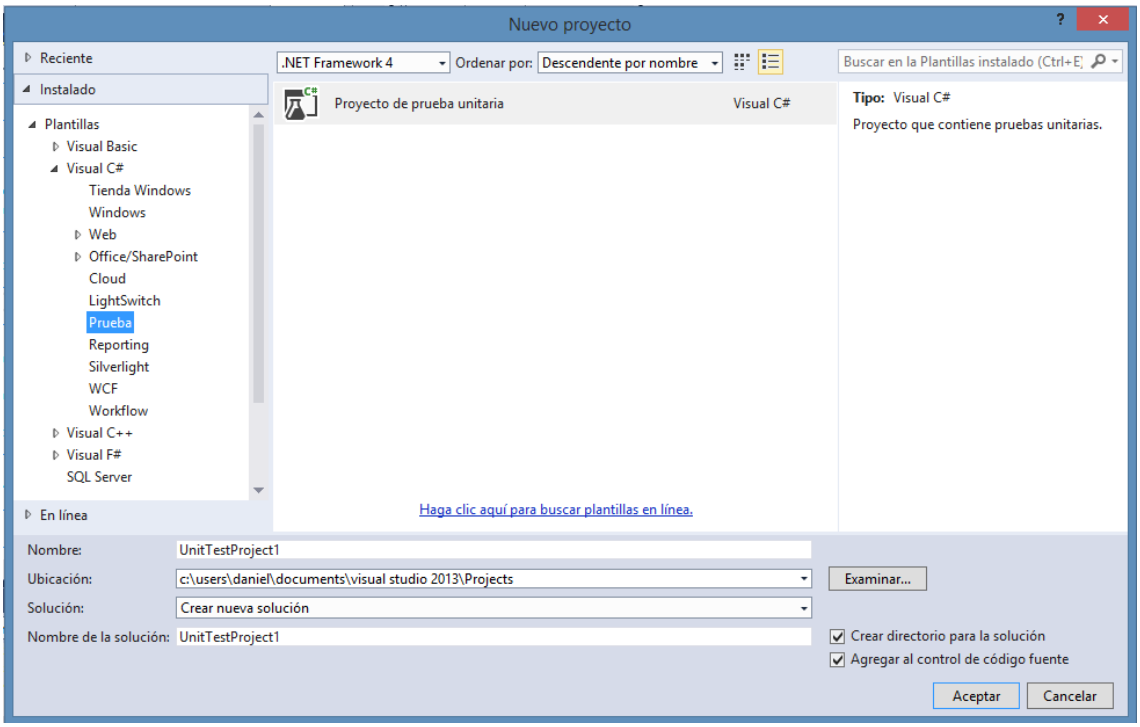


Ilustración 17 Pruebas Unitarias.

Dentro del proyecto nos creara una clase de *UnitTest* con una prueba unitaria vacía.
El atributo [*TestClass*] se utiliza para indicar al compilador que se trata de una clase de prueba mientras que el atributo [*TestMethod*] indica a que un determinado método es una prueba unitaria. Es importante destacar que las pruebas unitarias deben devolver *void* y no tener parámetros.

Vamos a ver un ejemplo de prueba unitaria dada una expresión regular para probar el método analize con la expresión regular de un DNI: \b\d{8}[A-Z]\b

```
[TestClass]
public class UnitTest1
{
    [TestMethod]
    public void DNI()
    {
        Analizador ana = new Analizador();
        RegExItem resultado = ana.analize("\\b\\d{8}[A-Z]\\b", true);
        RegExItem esperado = new RegExItem();
        esperado.Tipo = 0;
        esperado.Subtipo = 0;
        esperado.Componentes = new System.Collections.ArrayList();
        RegExItem primero = new RegExItem("Limite de Palabra",1,257,"");
        esperado.Componentes.Add(primero);
        RegExItem segundo = new RegExItem("Digito",1,255,"b8");
        esperado.Componentes.Add(segundo);
        RegExItem tercero = new RegExItem(null,4,2,"");
        RegExItem tercero1 = new RegExItem("A-Z",9,0,null);
        tercero.Componentes.Add(tercero1);
        esperado.Componentes.Add(tercero);
        esperado.Componentes.Add(primero);
        Assert.AreEqual(esperado, resultado);
    }
```

Ilustración 18 Código para una Prueba Unitaria.

Esta prueba nos asegura que dada la expresión regular de un DNI el resultado es el correcto.

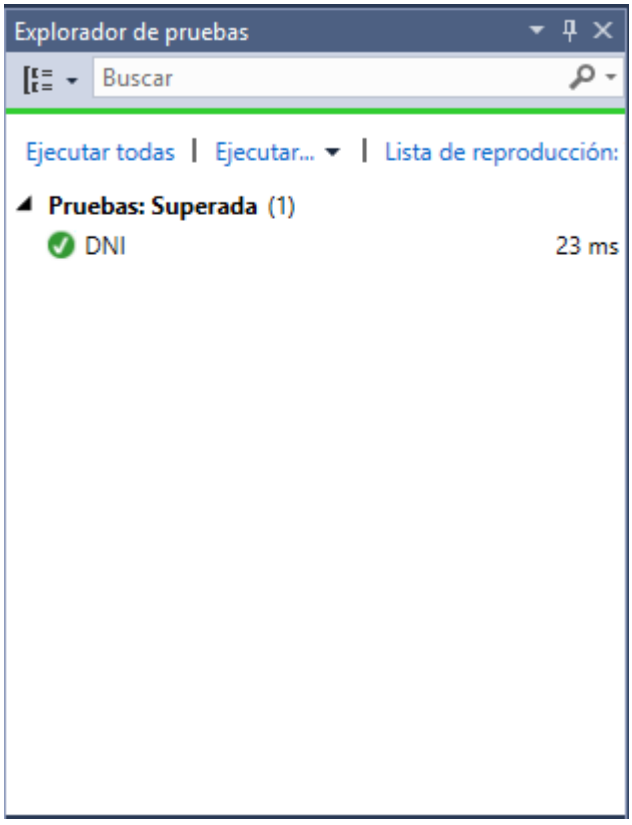


Ilustración 19 Ejemplo de Prueba Superada.

Otro de los puntos a probar en la aplicación web es usando una expresión regular probamos introducir un texto para realizar una búsqueda de algún patrón que cumple con la expresión regular. En la siguiente imagen se puede observar que existe una ocurrencia en dicho texto marcado en rojo y además existe un único grupo marcado en azul. Además también se puede comprobar que la interfaz está en el idioma ingles para de esta forma demostrar que la aplicación web esta internacionalizada.

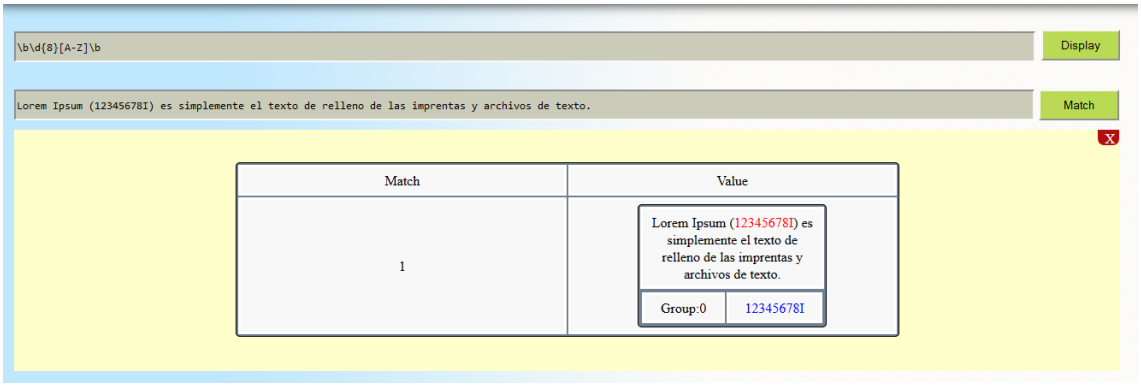


Ilustración 20 Resultado de buscar ocurrencias dentro de un texto.

7.-Resultados y Conclusiones.

Un aspecto fundamental a la hora de dar por finalizado un Trabajo de Fin de Grado es comprobar que los objetivos marcados en el comienzo del mismo se han satisfecho con el trabajo desarrollado.

En este caso se han podido cumplir todos los objetivos propuestos al inicio del proyecto. Se ha conseguido realizar una aplicación web que permite obtener un Autómata Finito dado una expresión regular, además de las búsquedas de patrones en textos. Todo ello gestionado por un Servicio Web basado en *ASP.NET*.

Gracias a la realización de este proyecto se han podido estudiar nuevas tecnologías y librerías de programación, debido a la necesidad de integrar lenguajes de programación de distinta índole. No solo ha sido necesario conocer los lenguajes de programación utilizados en el desarrollo de la herramienta, sino también las metodologías y estrategias. Debido a esta circunstancia el trabajo realizado para el desarrollo se ha enriquecido enormemente.

Por otro lado, el haber conseguido implementar una herramienta capaz de monitorizar las expresiones regulares, es muestra de que sin duda es un conocimiento que podrá ser utilizado en un futuro ya sea como empleado, o para emprender un proyecto personal de diferentes características. Y que además dicha herramienta esta internacionalizada, dado que puede adaptarse a diferentes idiomas y regiones sin la necesidad de realizar cambios de ingeniería ni en el código.

Sin lugar a dudas, la realización de un trabajo de fin de grado de estas características requiere una gran dedicación y esfuerzo, pero se ven recompensados por la satisfacción de haber conseguido desarrollar una herramienta completa desde cero que pueda ser utilizado por aquellos alumnos que están aprendiendo a programar y que necesitan del conocimiento de las expresiones regulares. Sin olvidar la gran cantidad de aspectos que se han aprendido durante todo el proceso.

8.-Trabajo Futuro.

En las matemáticas computacionales las expresiones regulares tienen como objetivo representar todos los posibles lenguajes definidos sobre un alfabeto Σ, en base a una serie de lenguajes primitivos, y unos operadores de composición.

Lenguajes primitivos: el lenguaje vacío, el lenguaje formado por la palabra vacía, y los lenguajes correspondientes a los distintos símbolos del alfabeto.

Operadores de composición: la unión, la concatenación y el cierre.

Durante la carrera en la asignatura de Matemáticas Computacionales explican cómo obtener un autómata o una gramática regular a partir de una expresión regular y viceversa. Como trabajo futuro se pueden añadir funcionalidades a este Proyecto que apliquen sobre una expresión regular el teorema de Thompson que sirve para obtener autómatas finitos no deterministas con transiciones vacías (*AFND-ε*), y una vez hecho esto seguir aplicando métodos para pasar de (*AFND-ε*) a Autómata Finito No Determinista (*AFND*), y de *AFND* a Autómata Finito Determinista (*AFD*). Que a su vez suele ocurrir que el *AFD* no es el mínimo y se apliquen otros algoritmos para minimizar. [3].

Además debido a que quedara un Servicio Web ya programado, también como trabajo futuro se puede realizar una aplicación móvil diseñada para ser ejecutada en todos los dispositivos móviles (Tablet, teléfonos inteligentes).

9.-Bibliografía.

1. Internet Information Services-Wikipedia, la enciclopedia libre (En línea): http://es.wikipedia.org/wiki/Internet_Information_Services (10/10/2014).
2. Expresión regular - Wikipedia, la enciclopedia libre (En línea): http://es.wikipedia.org/wiki/Expresi%C3%B3n_regular (20/09/2014).
3. Algoritmo de Thompson - Wikipedia, la enciclopedia libre (En línea): http://es.wikipedia.org/wiki/Algoritmo_de_Thompson (15/10/2014).
4. Internacionalización y localización - Wikipedia, la enciclopedia libre (En línea): http://es.wikipedia.org/wiki/Internacionalizaci%C3%B3n_y_localizaci%C3%B3n (05/10/2014).
5. Raphaël Reference (En línea): http://raphaeljs.com/reference.html (01/09/2014).
6. Cómo: Crear y ejecutar una prueba unitaria (En línea): http://msdn.microsoft.com/es-es/library/vstudio/dd286656(v=vs.100).aspx (25/10/2014).
7. JavaScript - Wikipedia, la enciclopedia libre (En línea): http://es.wikipedia.org/wiki/JavaScript (01/09/2014).
8. JQuery (En línea): http://jquery.com/ (05/09/2014).
9. Visual Studio - Herramientas para desarrolladores de Microsoft (En línea): http://www.visualstudio.com/ (01/08/2014).