UNIVERSIDAD POLITÉCNICA DE MADRID

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo fin de grado

¿Es tu diapositiva de fácil lectura?

***Autor**: Sandra Cartas Sánchez*
***Tutor**: Mari Carmen Suárez de Figueroa Baonza*

Curso académico: 2016/2017

Semestre: Febrero – Junio

Agradecimientos

A mi tutora por brindarme la oportunidad de realizar este trabajo que ha supuesto una gran satisfacción personal.

A mis padres y mi hermana por el apoyo incondicional desde que empezó esta aventura hace unos años, por confiar en mi plenamente, por sus consejos y por estar siempre dispuestos a ayudarme en todo.

A mi pareja, porque ha supuesto mi mayor apoyo y me ha ayudado a superarme cada día. Por darme confianza, por su infinita paciencia y empujarme a seguir en todo momento.

A mis compañeros de carrera porque no puede haber un grupo mejor que el que formamos, gracias por los momentos épicos, por los julios de monsters hasta reventar y por nuestros grupos de estudios claves para finalizar la carrera. Sois geniales CRPNS

Tabla de contenido

Índice ilustraciones

Índice tablas

Resumen

Este proyecto tiene como objetivo desarrollar un servicio web que sea capaz de analizar y ponderar una presentación en formato XML en base a los estándares de la metodología Lectura Fácil. Para ello, se utilizará el lenguaje de programación Ruby con el framework Rails que sigue el paradigma Modelo Vista Controlador

Abstract

This Project has the aim of put into practice a web service that be able to analyse and weigh a slide in XML format, in base of the methodology's standars: Easy Reading. Therefore, will be used the language of programming Ruby with the Framework Rails who follows the paradigm MVC.

1. Introducción

La finalidad de este primer capítulo es proporcionar al lector una visión general de este proyecto, explicando su contexto, motivación, los objetivos planteados y la forma en que se espera que sean satisfechos. De igual forma se explica la estructura del resto del documento.

1.1 Contexto

Este proyecto se enmarca dentro del área de la accesibilidad de la información aplicada a las presentaciones. En primer lugar, se va a comentar una breve introducción sobre las presentaciones o diapositivas utilizadas en el ámbito docente.

Una diapositiva es un documento informático que puede incluir textos, imágenes, objetos gráficos, fotografías, sonidos, animaciones, fragmentos de vídeo, hipervínculos … y que puede verse en la pantalla del ordenador.

Respecto a este ámbito docente, los usos de materiales de apoyo visual son siempre necesarios y eficaces para la docencia. Existen diversos recursos gráficos para ofrecer este apoyo visual, pero lo más utilizado son las presentaciones en Power Point(diapositiva). Hay que señalar que el uso de una diapositiva es simplemente un recurso, un apoyo para el docente, en ningún caso el docente se debe limitar a leer la presentación como si esta fuera un papel que contiene toda la información que quiere exponer de manera literal.

Como se ha mencionado esta presentación carece de sentido sin un expositor que explique el contenido de la misma. Por ello el objetivo principal de estas diapositivas es concentrar en ellas las ideas importantes y el expositor (en este caso el docente) es quien tiene que profundizar en estas ideas y darle sentido a la diapositiva.

Por todo esto el factor más importante de una diapositiva es el diseño de la misma, ya que este puede influir tanto positiva como negativamente en el receptor. Por ejemplo, una diapositiva saturada de texto se hace visualmente estresante y provocará rechazo en los oyentes, también causará desinterés y falta de atención por parte de los estudiantes. Otro ejemplo sería el uso de unos colores demasiado fuertes tanto en el fondo como en los textos.

Una diapositiva mal diseñada se caracteriza por tener una desorganización en el contexto de la presentación causando una dificultad en el proceso de aprendizaje.

Las diapositivas mal diseñadas suelen usar unos colores demasiado fuertes tanto en el fondo como en los textos, esto genera distracción en los oyentes. Otro factor es la cantidad de texto por diapositiva, si la diapositiva contiene una gran cantidad de texto el público leerá la diapositiva

por lo que no prestará atención al ponente. Por último, mencionar la falta de contraste entre el fondo y el texto.

Para realizar una buena diapositiva existen una serie de 'buenas prácticas' para hacer de tu diapositiva un elemento de información perfecto.

Los elementos claves de una presentación son:

- Diseño
- Organización
- Contenido

(Referencias [2], [3] de la bibliografía)

A continuación, se exponen varias pautas a tener en cuenta a la hora de diseñar una diapositiva para conseguir que esta sea un éxito.

- Es necesario simplificar. Las diapositivas han de ser siempre sencillas, simples y sin información que será irrelevante. Tampoco es aconsejable el uso de efectos especiales en la diapositiva simplemente por el hecho de llamar la atención o porque quede visualmente más atractivo. Si se emplean dichos efectos debe de hacerse porque hay buenas razones psicológicas, didácticas o pedagógicas detrás.

- Utilizar textos cortos y directos, gráficas fáciles de comprender e ilustraciones que reflejan lo que el profesor expresa en cada momento. Incluso que los textos no utilicen más de cinco palabras por línea ni más de cinco líneas por diapositiva. A esto se refiere Keer cuando advierte *"No estropee su trabajo con una saturación de texto y gráficos. Pregúntese ¿realmente es necesario todo lo que aparece en la pantalla?"*.

- Espacios en blanco. Es recomendable que la diapositiva contenga espacios 'en blanco'. Evitar llenar la diapositiva con varios logos, tablas o gráficos y menos aún de texto sobre todo si en nada contribuye al logro de los propósitos de la clase. Tampoco es conveniente el uso de diapositivas prediseñadas porque sean visualmente más atractivas.

- Es importante ser visual. El uso de imágenes y fotografías es necesario ya que constituye un medio poderoso de comunicación humana al reforzar el tema tratado y porque genera sentimientos y estados de ánimo favorables en los estudiantes y los motiva. Sin embargo, es importante cuidar que las imágenes usadas sean de calidad y sobre todo que estén directamente relacionadas con el tema que se está exponiendo.

- Carácter instrumental. El apoyo visual que proporciona la diapositiva nunca debe convertirse en el centro de atención de la exposición. La diapositiva es un apoyo para el profesor no la clase en sí misma. Los estudiantes no deben ver simplemente una sucesión de diapositivas, ellos necesitan escuchar al docente para así poder comprender las mismas y así afianzar la información que esta ofrece.

- Colores contrastados. Un buen contraste entre el fondo y el texto es muy importante a la hora de transmitir el mensaje. Se deben emplear los colores con conocimiento y siempre eligiendo los colores que contrasten entre ellos.

- Relevancia de lo expuesto. Los elementos visuales más eficaces son aquellos que pueden entenderse a simple vista. Para ello estos elementos han de ser claros y sencillos. Además, estos elementos tienen que satisfacer la necesidad de los alumnos para ayudarles a entender el tema que se expone.

(Referencia [1] de la bibliografía)

Además de seguir estas 'buenas prácticas' es necesario plantearse antes de realizar una diapositiva, a qué público va a ir dirigida la presentación, ya que el público final pueden ser personas que tengan algún tipo de minusvalía y esto ha de tenerse en cuenta para diseñar una buena diapositiva que llegue de una forma directa al receptor y que esta esté adaptada perfectamente a sus necesidades.

Este proyecto se centra en el desarrollo de un servicio web que sea capaz de validar que tan óptima es una diapositiva en base a unos estándares fijados para un público con capacidad intelectual reducida. A continuación, se tratará este tema en profundidad.

1.1.1 Discapacidad intelectual

La discapacidad intelectual es un término que se utiliza para describir a aquellas personas con ciertas limitaciones a la hora de aprender a leer, hablar, caminar y las destrezas del cuidado personal. Estas limitaciones pueden ocasionar que un niño se desarrolle y aprenda de una forma más lenta o de una forma diferente que un niño que se desarrolla de una forma normal. La discapacidad intelectual puede dar la cara antes de que la persona llegue a los 18 años de edad incluso antes de nacer.

La discapacidad intelectual puede ser causada por una lesión, enfermedad o un problema en el cerebro Para muchas personas la causa de la discapacidad intelectual es desconocida.

Las causas más frecuentes de la discapacidad intelectual son:

- Condiciones genéticas
- Complicaciones durante el embarazo
- Problemas durante el nacimiento
- Enfermedades o exposición a tóxicos

Esta discapacidad se caracteriza porque, aunque un grupo de personas sufren esta discapacidad cada una lo tendrá en un grado diferente. Por ejemplo, una persona puede tener mayor déficit de memoria y otra persona puede que tenga un grado de orientación muy bajo etc. Debido a esto es difícil realizar unos estándares para poder facilitar la accesibilidad a un amplio grupo de personas. Pero sí que se pueden dar soluciones parciales para determinadas capacidades.

(Referencias [4], [5], [6], [7], [9], [10], [11] de la bibliografía)

Este proyecto se centra en el campo de la educación, del aprendizaje a través de un medio visual como son las diapositivas (diapositiva). Como este proyecto se basa en el análisis de una diapositiva la capacidad a la que está directamente ligado es la comprensión lectora. Este servicio web se apoyará en la solución parcial conocida como 'Lectura Fácil'.

1.1.2 Lectura Fácil

La lectura fácil surge como una herramienta de compresión lectora y de fomento de la lectura para atraer a personas que no tienen el hábito de leer o que simplemente se han visto privadas de él.

En concreto esta herramienta busca facilitar el acceso a la información, la cultura y la literatura, debido a que es un derecho fundamental de las personas independientemente de sus capacidades.

Ilustración 1 - Logo de Lectura Fácil

Aunque hablamos de que es un derecho fundamental realmente las estadísticas señalaban que el 20% de los adultos del mundo no sabían ni leer ni escribir. Sin embargo, las cifras se elevan hasta casi un 30% si añadimos a los ciudadanos que, aunque saben leer y escribir no son capaces de leer textos complejos o incluso noticias.

(Referencia [8] de la bibliografía)

La Federación Internacional de Asociaciones de Bibliotecarios y Bibliotecas (IFLA) publicó en 1997 las "Directrices para materiales de lectura fácil" en ella se recogen dos ideas de lectura fácil.

- Adaptación lingüística de un texto que lo hace más fácil de leer que un texto medio, pero no es más fácil de comprender.
- Adaptación que permite una lectura y una comprensión más sencilla.

La solución parcial que proporciona la lectura fácil sólo será válida para personas que tengan capacidad de lectoescritura. Esta metodología ofrece una alternativa adecuada cuando hasta ahora la única posibilidad de acceso a la lectura que tenían muchos adultos con dificultades lectoras eran los libros para niños.

A continuación, se detallará el manual de buenas prácticas para considerar un texto como lectura fácil. Únicamente se van a mencionar aquellos apartados que guardan relación con el alcance del proyecto que estamos exponiendo.

TIPOGRAFÍA

- Utilizar dos tipos de letra como máximo: para texto y para títulos.
- El tamaño de letra debe ser suficientemente grande, entre 12 y 16 puntos, siendo una opción habitual 14 puntos, aunque varía según el tipo de letra.
- Utilizar tipografías sin remate, por ser más claras. Entre otras, se pueden destacar Arial, Calibri, Candara, Corbel, Gill Sans, Helvética, Myriad, Segoe, Tahoma, Tiresias y Verdana. Evitar caracteres ornamentados o que simulen la letra manuscrita.
- No utilizar caracteres muy finos, ni cursivas ni mayúsculas (en este último caso, salvo excepciones muy concretas en que sólo haya una o dos palabras).
- Utilizar negritas y subrayados para destacar palabras, aunque siempre de forma moderada para evitar distracciones. El uso del subrayado puede ser muy útil para destacar nombres de personas y lugares y facilitar la memorización.
- Evitar efectos tipográficos, como adornos, colores y sombras.
- Si se utilizan caracteres blancos, el fondo debe ser contrastado, aunque siempre la letra blanca sobre fondo de color es más difícil de leer que la letra negra sobre fondo blanco.
- Utilizar un interlineado acorde a la tipografía, que puede variar desde sencillo a más amplio (1,3 a 1,5 veces
- Mayor que el espacio medio entre palabras o 30% del tamaño de la letra). No es conveniente un interlineado demasiado amplio, porque la separación excesiva de líneas puede confundir.

Distribución del espacio

- Crear una distribución ordenada y poco densa del espacio en la página.
- El número de líneas por página será limitado. Es conveniente no incluir demasiada información en cada página. Una nueva idea se expondrá en una nueva página.
- Cuanto más blanco, mejor, pero siempre con texto en la página: utilizar amplios márgenes, blancos amplios en torno a los párrafos y líneas en blanco para distinguir párrafos y separar ideas.
- Evitar un diseño en columnas. Es mejor una composición horizontal que vertical. No obstante, si se incluyen, las columnas deben estar separadas claramente o con reglas.

(Referencia [11] de la bibliografía)

2. Objetivos del proyecto

El objetivo general del presente proyecto es desarrollar un servicio web que sea capaz de analizar y ponderar una presentación en formato XML en base a los estándares de la metodología Lectura Fácil. Para ello, se utilizará el lenguaje de programación Ruby con el framework Ruby on Rails que sigue el paradigma Modelo Vista Controlador. Los objetivos específicos de cara al desarrollo e implementación del servicio web en cuestión:

1. Estudio del desarrollo de servicios web con Ruby on Rails, CSS, Java (Figura 2) Script, JQuery y Bootstrap para conseguir el nivel necesario para desarrollar este servicio web.

2. Estudio del Modelo Vista Controlador utilizado por Ruby on Rails.

3. Estudio y entendimiento del funcionamiento de una base de datos POSTGRESQL

4. Diseño de un servicio web que permita analizar el código XML de una diapositiva (slide) en base a los estándares que sigue la metodología Lectura Fácil. Este servicio web también tiene que puntuar dicha diapositiva en base al número de estándares que cumpla, así como ofrecer en un informe que contenga información sobre la mejora de aquellos estándares que no cumple dicha diapositiva.

5. Realización de pruebas del software tanto durante el desarrollo del servicio web, así como al final de cada módulo.

   a) Pruebas Unitarias
   b) Pruebas de Integración
   c) Pruebas de Sistema
   d) Pruebas de Aceptación

6. Redactar memoria final. Este punto es muy importante de cara al desarrollo futuro de este servicio por otros desarrolladores.

También entran dentro de los objetivos específicos de este proyecto la realización de un estudio y análisis de la metodología de Lectura Fácil y conocer cuáles son los principales problemas que las personas con dificultades de comprensión lectora se encuentran a la hora de poder comprender un texto para poder extrapolar esta información y aplicarla al análisis de la diapositiva que pretendemos analizar.

3. Estado del Arte

El objetivo de este capítulo es presentar aquellas tecnologías y trabajos relacionados con el proyecto que se está exponiendo. Esto tiene como finalidad recopilar todas las ideas, conceptos opiniones o trabajos ya desarrollados para tomarlos como base a la hora de desarrollar este proyecto.

Este proyecto tiene como objetivo como se ha mencionado en otras ocasiones a lo largo de este documento, el desarrollo de un servicio web que sea capaz de validar y ponderar una diapositiva en base a los estándares que proporciona la metodología de lectura fácil. Por ello a lo largo de este capítulo se abordará el estudio y evaluación de las diversas tecnologías existentes, su estado actual y el porqué de su utilización frente a otras para el desarrollo de este servicio.

3.1 Situación actual del desarrollo de aplicaciones

Ahora nos encontramos en una situación donde el diseño y desarrollo de software web está en pleno crecimiento. Cada vez son más las empresas que optan por desarrollar un sistema software ya sea para gestionar de forma interna la empresa o también para darse a conocer y obtener beneficios a través de ellas.

Las aplicaciones web pueden servir para obtener información de clientes, generación de documentos necesarios, consultas de acciones realizadas por la empresa, pedidos, facturaciones, tareas a desarrollar, visualización de estadísticas, etc. Infinidad de posibilidades de acceso a la información desde cualquier punto, gracias a la evolución del software y la introducción de Internet.
Durante los últimos años ha ido modificándose el proceso de desarrollo de las mismas, ahora surgen aplicaciones dinámicas, basadas en la constante actualización.

Poco a poco, las aplicaciones Web se están convirtiendo en una funcionalidad más completa, mientras que están siendo fáciles de usar. Podemos tomar por ejemplo de Google Docs, Office Web Apps, BitDefender entre muchas otras

El impacto de las aplicaciones Web sobre cómo operar un negocio, transmitir y recibir información, e incluso en la vida de las personas es considerable. Las aplicaciones Web ofrecen la oportunidad de conectar a los usuarios entre sí y las empresas con sus clientes.

En resumen, podemos decir que el desarrollo de apps es un mercado joven todavía y que sigue en continuo crecimiento. Eso es lo que apuntan los datos recogidos sobre la actividad de los desarrolladores durante 2016 y que parecen intuir la tendencia para este 2017.

3.2 Lenguajes de programación y entornos de desarrollo

Ahora es el momento de estudiar qué herramientas son las más apropiadas para el desarrollo de dicho servicio web, así como el lenguaje de programación más conveniente para su desarrollo.

3.2.1 Java & Eclipse

El lenguaje de programación Java (Figura 2) es un lenguaje orientado a objetos de una plataforma independiente. Ese lenguaje fue desarrollado por la compañía Sun Microsystems, con la idea original de usarlo para la creación de páginas web. Java (Figura 2) tiene muchas similitudes con los lenguajes C y C++, así que, si se tiene conocimiento de este lenguaje, el aprendizaje de la programación en Java (Figura 2) será de fácil comprensión para un programador que ya haya realizado programas en estos lenguajes.

Ilustración 2 - Logo de Java

Con la programación en Java (Figura 2) se pueden realizar tantos applets que son unas aplicaciones especiales que se ejecutan dentro de un navegador al ser cargada una página HTML en un servidor web. Otra utilidad es el desarrollo de aplicaciones que finalmente son programas independientes, es decir, JAVA se puede utilizar por ejemplo para procesar palabra, realizar una hoja de cálculo o un sistema de gestión para una empresa.

En resumen, cualquier tipo de aplicación se puede realizar con este lenguaje.
Java (Figura 2) permite el modularidad por lo que se pueden hacer métodos o clases individuales que pueden ser usados por más de una aplicación.

La programación en Java (Figura 2), permite el desarrollo de aplicación siguiendo un esquema Cliente - Servidor, como de aplicaciones distribuidas, lo que hace capaz de conectar dos o más ordenadores ejecutando tareas de forma simultánea y de esta forma logra distribuir el trabajo que tiene que realizar.

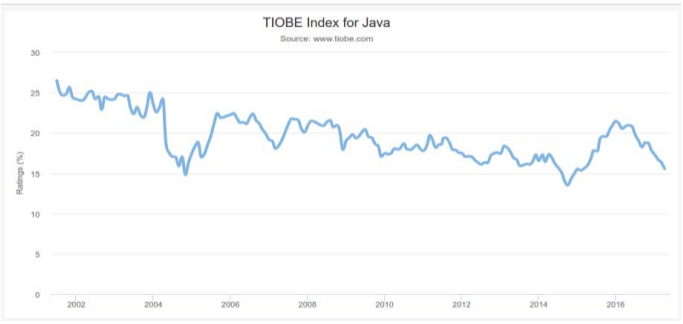


Ilustración 3 - Popularidad del lenguaje de programación JAVA. (Fuente: TIOBE)

(Referencias [13], [15], [16], [21] de la bibliografía)

Eclipse es una plataforma de desarrollo diseñada para ser extendida de forma se podría decir que indefinida a través de plugins. Eclipse (Figura 4) no está diseñado para un lenguaje en específico, sino que es un IDE genérico, aunque si bien es cierto tiene mucha popularidad en la comunidad de desarrolladores del lenguaje JAVA.

Ilustración 4 - Logo de Eclipse

Este entorno proporciona herramientas para la gestión de los espacios de trabajo, escribir, desplegar, ejecutar y depurar las aplicaciones.

Como principales características cabe destacar las perspectivas, editores y vistas. En eclipse el concepto de trabajo está basado en las perspectivas que son una preconfiguración de ventanas y editores relacionadas entre sí que permiten realizar el trabajo en un determinado entorno de forma óptima.

También hay que mencionar la gestión de proyectos ya que Eclipse se basa en los proyectos que son un conjunto de recursos que están relacionados entre sí como por ejemplo puede ser el código fuente y la documentación o archivos de configuración, imágenes etc.

Una de las características más importante de Eclipse es su sistema de depuración del código ya que es de fácil uso y bastante intuitivo además visualmente ayuda al programador a mejor el código.

Para finalizar se menciona la extensa cantidad de colecciones de plugin que están disponibles ya sea por Eclipse o por terceros.

3.2.2 C# & Microsoft Visual Studio

C# es un lenguaje de programación creado por Anders Hejlsberg al igual que los lenguajes Turbo Pascal y Delphi. C# es un lenguaje de programación orientado a objetos. Este lenguaje se diseñó con el propósito de mejorar con respecto de los dos lenguajes anteriores.

En este nuevo lenguaje se quiso incorporar las ventajas o mejorar que tiene el código JAVA. Con esto se consiguió que este lenguaje tuviese lo bueno de C y de C++, pero además la productividad que posee el lenguaje JAVA.

Como características a destacar de ese lenguaje se debe mencionar la posibilidad de tratar íntegramente su código como un objeto. También tiene ventaja que la sintaxis de este nuevo lenguaje es muy similar a la de JAVA. Es un lenguaje orientado a objetos y a componentes o módulos. Además, se ahorra mucho tiempo en la programación ya que posee una librería de clases muy completa y bien diseñada. Aunque C# forma parte de la plataforma .NET, es un lenguaje totalmente independiente.

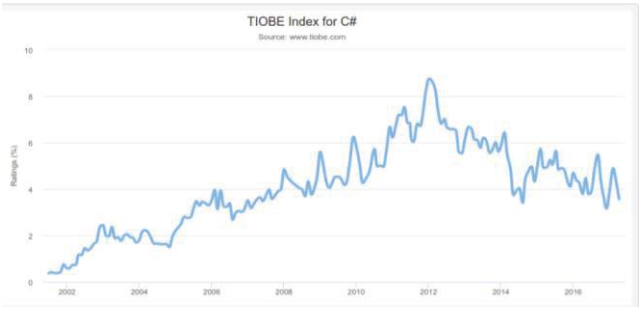


Ilustración 5 - Popularidad del lenguaje de programación C#. (Fuente: TIOBE)

(Referencias [12], [15], [17], [20] de la bibliografía)

Microsoft Visual Studio es un Entorno de Desarrollo Integrado (IDE) para el diseño de aplicaciones y servicios para sistemas operativos Windows. Se pueden programar en numerosas sintaxis: C++, C#, Visual Basic .NET, Python, Ruby, PHP o Java (Figura 2). También en entornos de desarrollo web como ASP. NET MVC (Modelo Vista Controlador) o Django. Lo que se buscaba conseguir con este entorno es una programación multilenguaje, es decir que te permita realizar diferentes desarrollos en diferentes lenguajes en el mismo entorno de programación.

Ilustración 6 - Logo de Visual Studio

Profundizando un poco en el desarrollo de aplicaciones con Visual Studio (Figura 6) cabe señalar que permite realizar el desarrollo de estas aplicaciones con diferentes opciones.

Permite el diseño de aplicaciones nativas e híbridas, también diseño de aplicaciones nativas en lenguaje C#, desarrollo de aplicaciones nativas con C++ y desarrollo de aplicaciones híbridas en Java (Figura 2) Script entre otras.

Además, Visual Studio está dotado de herramientas para codificar, depurar y probar apps.

(Referencia [20] de la bibliografía)

3.2.3 Ruby & Atom (editor de textos)

Ruby (Figura 7) fue creado en Japón por Yukihiro Matsumoto mientras trabajaba como programador con lenguajes como Perl y PHP. La intención de esta persona fue la de crear un Perl avanzado debido a que deseaba mejor algunas de las particularidades de este lenguaje. Pero en lugar de mejorarlo decido desarrollar uno propio a partir de lenguajes como Perl, Smalltalk, Eiffel y Lisp.

Ilustración 7 - Logo de Ruby

Se destaca una frase que dijo Yukihiro en una charla de tecnología en Google: *"I hope to see Ruby help every programmer in the world to be productive, and to enjoy programming, and to be happy. That is the primary purpose of Ruby language."*

Ruby se presenta como un lenguaje sencillo y flexible y por esta razón atrae a programadores de todos los sectores. Ruby es un lenguaje multipropósito que permite desarrollos en las siguientes áreas.

- Aplicaciones comerciales
- Acceso a base de datos
- Proceso y transformación de XML
- Aplicaciones distribuidas
- Aplicaciones web

Ahora se van a detallar una serie de características muy importantes a destacar sobre este lenguaje.
Ruby (Figura 7) es un lenguaje de scripts es moderno y está orientado a objetos. Incorpora algunas de las mejores características de lenguajes como Smalltalk, Java (Figura 2) y Perl. Respecto al diseño de aplicaciones tiene un alcance ilimitado ya que nos podemos encontrar desarrollos web hasta diseños complejos como simulaciones. Es un lenguaje multiplataforma que se integra perfectamente en gran cantidad de arquitecturas. Promueve las mejoras prácticas sin renunciar a la usabilidad.

(Referencias [14], [19] de la bibliografía)

Con el uso de Ruby se pueden complementar las características de la lógica imperativa con la funcional.
Posee una filosofía de trabajo DRY ('Don't repeat yourself; en castellano: No te repitas'), esto se implementa mediante sus conocidos 'helpers'. Simplifica la declaración de estructuras y modelos. Es un lenguaje dinámico e interpretado.

A continuación, se van a desglosar algunas de las características más internas del lenguaje:

- En Ruby todo es un objeto, esto quiere decir que hasta el más simple carácter es instancia de clase y serán manipuladas como tal.
- En este lenguaje todo tiene un valor, aunque sea nil.
- Se debe tener en cuenta que no existe en principio distinción entre comandos y expresiones dentro del lenguaje de programación.
- Ruby utiliza solo herencia simple, sin embargo, incorpora técnicas para poder imitar el comportamiento de la herencia múltiple de manera sencilla.
- Utiliza un recolector de basura de alto nivel. Por lo tanto, libera al desarrollador de estas tareas.
- No requiere la declaración de variables.
- Ruby permite la programación con múltiples hilos de formas independiente al sistema operativo.

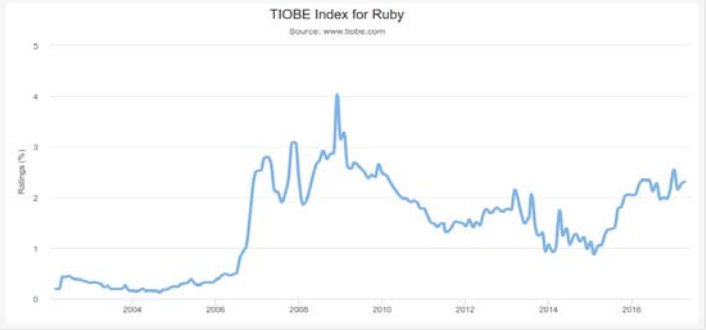


Ilustración 8 - Popularidad del lenguaje de programación Ruby. (Fuente: TIOBE)

(Referencias [14], [15], [19] de la bibliografía)

Atom es un editor de texto desarrollado por el equipo de GitHub cuya principal característica es la de ser un editor de texto fácilmente hackeable de manera que cualquier programador podrá modificar el código y así adaptar el editor a sus necesidades.

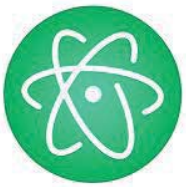


Ilustración 9 - Logo de Atom

Atom(Figura 9) está escrito totalmente en HTML, Java (Figura 2) Script, CSS y Node.js y se distribuye totalmente como 'código abierto'.

Lo mejor de este editor de texto es la cantidad de paquetes y plugins que tiene para extender su funcionamiento.

(Referencia [18] de la bibliografía)

Los lenguajes C# y Java (Figura 2) comparten ciertas similitudes en su sintaxis mientras que Ruby está algo más alejado de esa sintaxis. Cualquiera de las alternativas que se han expuesto sería una gran candidata para llevar a cabo el desarrollo del servicio web.

Finalmente, el lenguaje de programación seleccionada para desarrollar este servicio será Ruby (Figura 7) con el framework Ruby on Rails (esto se desarrollará más en profundidad en capítulos posteriores).

La decisión del porque este lenguaje frente al resto es por la experiencia en el desarrollo de aplicaciones web con este lenguaje y con el editor de textos Atom(Figura 9) antes comentado.

Además de por la experiencia, Ruby (Figura 7) ha sido la opción ganadora porque es muy sencillo y rápido crear la aplicación web. Mientras que en los otros lenguajes este proceso es más costoso y requiere más líneas de código Ruby con un par de comandos lanzados desde su consola ya te levanta el principio de tu aplicación web.

Otro punto a tener en cuenta es el sistema de gestión de la base de datos, Ruby puede desplegar la base de datos de tu aplicación en cualquier tipo de base de datos ya sea SQL Server, MySQL, PostgreSQL o cualquier otra gracias a su sistema de migraciones. Y por supuesto hay que mencionar la base de Ruby que son sus gemas que facilitan y agilizan la programación. Las gemas de Ruby son paquetes de librerías para Ruby que se instalan en el sistema y quedan listas para ser usadas.

3.3 Sistema de gestión de la base de datos

3.3.1 MySQL

MySQL es un sistema de gestión de bases de datos relacional, licenciado bajo la GPL de la GNU. Su diseño multihilo le permite soportar una gran carga de forma muy eficiente. MySQL (Figura 10) fue creado por la empresa sueca MySQL AB, que mantiene el copyright del código fuente del servidor SQL, así como también de la marca.

Ilustración 10 - Logo de MySQL

Este gestor de bases de datos es probablemente el más usado en el mundo del software libre, debido a su gran rapidez y facilidad de uso. Esta gran aceptación es debida a la existencia de infinidad de librerías y otras herramientas que permiten su uso a través de gran cantidad de lenguajes de programación, además de su fácil y cómoda instalación y configuración.

Las principales características de este gestor de bases de datos son:

1. Aprovecha la potencia de sistemas multiprocesador gracias a su implementación multihilo.
2. Soporta gran cantidad de tipos de datos para las columnas.
3. Dispone de API's en gran cantidad de lenguajes.
4. Gran portabilidad entre sistemas
5. Soporta hasta 32 índices por tabla.
6. Gestión de usuarios y contraseñas manteniendo un buen nivel de seguridad en los datos.

Por el contrario, carece de algunas características como:

1. No admite subconsultas.
2. SELECT INTO TABLE no está implementado.
3. Triggres y Procedures
4. Transacciones
5. Integridad referencial: aunque si admite la declaración de claves ajenas en la creación de tablas, internamente no las trata de forma diferente al resto de campos.

(Referencias [23], [25] de la bibliografía)

3.3.2 SQLServer

SQL Server es un sistema gestor de base de datos relacionales producido por Microsoft. Es un sistema cliente/servidor que funciona como una extensión natural del sistema operativo Windows. Entre otras características proporciona integridad de datos, optimización de consultas, control de concurrencia y backup y recuperación.
Es fácil de administrar a través de la utilización de un entorno gráfico para casi todas las tareas de sistema y administración de bases de datos.

Ilustración 11 - Logo SQL Server

Utiliza servicios del sistema operativo Windows para ofrecer nuevas capacidades o ampliar la base de datos, tales como enviar y recibir mensajes y gestionar la seguridad de la conexión. Es fácil de usar y proporciona funciones de almacenamiento de datos que sólo estaban disponibles en Oracle y otros sistemas gestores de bases de datos de mayor coste.

(Referencias [24], [25] de la bibliografía)

3.3.3 Postgresql

PostgreSQL es un sistema de gestión de base de datos objeto-relacional basado en el proyecto POSTGRES, de la universidad de Berkeley. El director de este proyecto es el profesor Michael Stonebraker y fue patrocinado por Defense Advanced Research Projects Agency.
Es un sistema objeto-relacional, ya que incluye características de la orientación a objetos, como puede ser la herencia, el tipo de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. Aunque a pesar de todo esto no es considerado un sistema de gestión de bases de datos puramente orientado a objetos.

Ilustración 12 - Logo PostgreSQL

A continuación, se enumeran las principales características de este gestor de bases de datos:

1. Implementación del estándar SQL92/SQL99.
2. Soporta distintos tipos de datos y permite la creación de tipos propios.
3. Incorpora una estructura de datos array.
4. Incorpora funciones: manejo de fechas, geométricas, orientadas a operaciones con redes, etc.
5. Permite la declaración de funciones propias, así como la definición de disparadores.
6. Soporta el uso de índices, reglas y vistas.

(Referencias [22], [25] de la bibliografía)

Finalmente se ha decidido usar PostgreSQL como gestor de base de datos.

Esta decisión ha sido en base a que una vez realizado el estudio de los gestores disponibles, se cree que PostgreSQL es el que mejor se adapta a las necesidades de nuestro servicio web, ya sea en el caso de uso de las reglas o vistas y funciones. Además, esta decisión también está afianzada por la experiencia de varios años en el manejo de este gestor de base de datos

A continuación, se detallan el resto de tecnologías y herramientas que serán utilizadas en la implementación del sistema:

3.4 Tecnologías y herramientas

3.4.1 GitHub

GitHub aloja tu repositorio de código y te brinda **herramientas** muy útiles para el **trabajo en equipo**, dentro de un proyecto.

En la actualidad, GitHub es mucho más que un servicio de alojamiento de código. Además de éste, se ofrecen varias herramientas útiles para el **trabajo en equipo**. Entre ellas, caben destacar:

- Una wiki para el mantenimiento de las distintas versiones de las páginas.
- Un sistema de seguimiento de problemas que permiten a los miembros de tu equipo detallar un problema con tu software o una sugerencia que deseen hacer.
- Una herramienta de revisión de código, donde se pueden añadir anotaciones en cualquier punto de un fichero y debatir sobre determinados cambios realizados en un commit específico.
- Un visor de ramas donde se pueden comparar los progresos realizados en las distintas ramas de nuestro repositorio.

3.4.2 Heroku

Heroku es una plataforma donde los desarrolladores pueden desplegar sus aplicaciones (sobretodo está pensado para aplicaciones web) y hacerlas públicas de forma gratuita o por un pequeño precio
Heroku es una plataforma en la nube para hosting y explotación de aplicaciones de lado de servidor. Con Heroku te olvidas de configurar máquinas Linux, servidores web y temas más avanzados como balanceo de carga, escalabilidad, etc. Especificando algunas dependencias y configuración básica de la BBDD vas a tener tu código en producción cada vez que lo desees sin más que hacer un push usando Git.

3.4.3 HTML

HTML es un lenguaje de marcado de hipertexto, básicamente este lenguaje se escribe en su totalidad con elementos, estos elementos están constituidos por etiquetas, contenido y atributos.

HTML es un lenguaje que interpreta el navegador web para mostrar los sitios o aplicaciones web tal y como estamos acostumbrados.
Las etiquetas nos sirven para delimitar el inicio y el fin de un elemento, como en el ejemplo, vemos un elemento que utiliza la etiqueta de apertura

3.4.4 JavaSript

JavaScript, es uno de los más potentes e importantes lenguajes de programación en la actualidad, por tres enfoques claros: es útil, práctico y está disponible en cualquier navegador web.

Características:

- Es Liviano.
- Multiplataforma, ya que se puede utilizar en Windows, Linux o Mac o en el navegador de tu preferencia.
- Es estructurado, mediante un conjunto de instrucciones indica al computador qué tarea debe realizar.
- Orientado a objetos y eventos.
- Es Interpretado, no se compila para poder ejecutarse.

3.4.5 JQuery

JQuery es simplemente una librería específica de código JavaScript. Existen muchas otras librerías JavaScript como MooTools, pero jQuery se ha convertido en la más popular debido a su facilidad de uso y su gran potencia.
Por lo tanto, Jquery no es un lenguaje distinto de Javascript, es simplemente una librería que complementa a JavaScript y mejora cierta de sus funcionalidades.

3.4.6 Boostrap

Bootstrap es una excelente herramienta para crear interfaces de usuario limpias y totalmente adaptables a todo tipo de dispositivos y pantallas, sea cual sea su tamaño. Además, Bootstrap ofrece las herramientas necesarias para crear cualquier tipo de sitio web utilizando los estilos y elementos de sus librerías.

Pricipales características:

- Soporte bastante bueno (casi completo) con HTML5 y CSS3, permitiendo ser usado de forma muy flexible para desarrollo web con unos excelentes resultados.

- Boostrap 3 establece Media Queries para 4 tamaños de dispositivos diferentes variando dependiendo del tamaño de su pantalla, estas Media Queries permiten desarrollar para dispositivos móviles y tablets de forma mucho más fácil.

- Boostrap 3 también permite insertar imágenes responsive, es decir, con solo insertar la imagen con la clase “img-responsive” las imágenes se adaptarán al tamaño.

- Boostrap es compatible con la mayoría de navegadores web del mercado, y más desde la versión 3, actualmente es totalmente compatible con los siguientes navegadores:
- Google Chrome (en todas las plataformas).
- Safari (tanto en iOS como en Mac).
- Mozilla Firefox (en Mac y en Windows).
- Internet Explorer (en Windows y Windows Phone).
- Opera (en Windows y Mac).

El diseño de la aplicación web está basado en la arquitectura Cliente/Servidor, esta arquitectura es una de las más utilizados para este tipo de sistemas. En los siguientes apartados se detallará una descripción acerca de esta arquitectura, así como los aspectos más importantes de ella.

3.4.7 Arquitectura Cliente/Servidor

La estructura cliente/servidor es una arquitectura en la que se consigue un procesamiento de forma cooperativa o conjunta de la información por medio de diversos procesadores trabajando en conjunto, de esta forma uno o varios clientes repartidos en diferentes zonas geográficas o no, solicitan servicio a uno o más servidores.

Por medio de esta arquitectura, todos los procesadores, clientes y servidores trabajan en ‘equipo’ es decir trabajar de forma cooperativa para realizar un tratamiento de la información.

Esta arquitectura consiste en que todas las tareas se reparten en los diferentes procesadores, los usuarios por su parte reciben el resultado de forma totalmente transparente. Aunque existan varios equipos(servidores) que han intervenido en el tratamiento de la información, de cara al usuario esto es totalmente transparente. Se puede decir por tanto que la arquitectura distribuida, posiblemente es la más extendida.

Elementos de la arquitectura Cliente/Servidor.

Un sistema Cliente/Servidor es un Sistema de Información distribuido basado en las siguientes características:

- Servicio: unidad básica de diseño. El servidor los proporciona y el cliente los utiliza.
- Recursos compartidos: Muchos clientes utilizan los mismos servidores y, a través de ellos, comparten tanto recursos lógicos como físicos.
- Protocolos asimétricos: Los clientes inician "conversaciones". Los servidores esperan su establecimiento pasivamente.
- Transparencia de localización física de los servidores y clientes: El cliente no tiene por qué saber dónde se encuentra situado el recurso que desea utilizar.
- Independencia de la plataforma HW y SW que se emplee.
- Sistemas débilmente acoplados. Interacción basada en envío de mensajes.
- Encapsulamiento de servicios. Los detalles de la implementación de un servicio son transparentes al cliente.
- Escalabilidad horizontal (añadir clientes) y vertical (ampliar potencia de los servidores).
- Integridad: Datos y programas centralizados en servidores facilitan su integridad y mantenimiento.

En el modelo usual Cliente/Servidor, un servidor, se activa y espera las solicitudes de los clientes.

Lo normal es que los servicios de un mismo servidor puedan ser utilizados por múltiples clientes distintos. Tanto los programas cliente como los servidores son con frecuencia parte de un programa o aplicación mayores.

Esquema Cliente/Servidor

El Esquema de funcionamiento de un Sistema Cliente/Servidor sería:

1. El cliente solicita una información al servidor.
2. El servidor recibe la petición del cliente.
3. El servidor procesa dicha solicitud.
4. El servidor envía el resultado obtenido al cliente.
5. El cliente recibe el resultado y lo procesa.

Componentes de la arquitectura Cliente/Servidor

- Nivel de Presentación: Agrupa a todos los elementos asociados al componente Cliente.
- Nivel de Aplicación: Agrupa a todos los elementos asociados al componente Servidor.
- Nivel de comunicación: Agrupa a todos los elementos que hacen posible la comunicación entre los componentes Cliente y servidor.
- Nivel de base de datos: Agrupa a todas las actividades asociadas al acceso de los datos.

Elementos principales

CLIENTE

Un cliente es todo proceso que reclama servicios de otro. Una definición un poco más elaborada podría ser la siguiente: *cliente es el proceso que permite al usuario formular los requerimientos y pasarlos al servidor*. Se lo conoce con el término front-end.

Las funciones que lleva a cabo el proceso cliente se resumen en los siguientes puntos:

- Administrar la interfaz de usuario.
- Interactuar con el usuario.
- Procesar la lógica de la aplicación y hacer validaciones locales.
- Generar requerimientos de bases de datos.
- Recibir resultados del servidor.
- Formatear resultados.

La funcionalidad del proceso cliente marca la operativa de la aplicación. De este modo el cliente se puede clasificar en:

- *Cliente basado en aplicación de usuario*. Si los datos son de baja interacción y están fuertemente relacionados con la actividad de los usuarios de esos clientes.
- *Cliente basado en lógica de negocio.* Toma datos suministrados por el usuario y/o la base de datos y efectúa los cálculos necesarios según los requerimientos del usuario.

SERVIDOR

Un servidor es todo proceso que proporciona un servicio a otros. Es el proceso encargado de atender a múltiples clientes que hacen peticiones de algún recurso administrado por él. Al proceso servidor se lo conoce con el término back-end. El servidor normalmente maneja todas las funciones relacionadas con la mayoría de las reglas del negocio y los recursos de datos. Las principales funciones que lleva a cabo el proceso servidor se enumeran a continuación:

- Aceptar los requerimientos de bases de datos que hacen los clientes.
- Procesar requerimientos de bases de datos.
- Formatear datos para transmitirlos a los clientes.

(Referencia [27] de la bibliografía)

4. Planteamiento del problema

Este apartado tiene como principal objetivo identificar y estipular cada una de las necesidades que el sistema final debe satisfacer. Para definir de forma clara y concisa cada una de estas necesidades y aportar una solución se realiza una especificación de requisitos.

Para realizar la fase de especificación de requisitos es necesaria la colaboración tanto de los ingenieros del sistema como de los propios usuarios. Como resultado de esta colaboración se define un estudio detallado de todas aquellas operaciones que el sistema debe de tener implementadas y se realizara un análisis de los casos de uso cada una de ellas.

Tras la realización de esta fase de especificación se obtiene los requisitos tanto funcionales como no funcionales y para cada uno de ellos su modelización en base a los casos de uso. Esta especificación permite que sea mucho más sencilla la compresión de aquellas necesidades que el sistema deberá cubrir para garantizar la satisfacción del cliente.

4.1 Definición del sistema

La idea principal que se presenta trata de la creación de una herramienta web por medio de la cual sea posible la evaluación de una diapositiva o slide (en formato XML) en función de una serie de estándares sobre Lectura Fácil. En base a esta evaluación la herramienta otorgará una puntuación a dicha diapositiva. Además, el sistema debe soportar llamadas mediante peticiones web para integrarse en otro módulo.

En la siguiente tabla se detallan los estándares que el sistema validará en la diapositiva de entrada.

LISTADO DE ESTÁNDARES

NÚMERO	TIPO	ESTÁNDAR	DESCRIPCIÓN
1	TIPOGRAFÍA	Fuentes utilizadas	Validar que el tipo de fuente del texto pertenece a los estilos aceptados
2	TIPOGRAFÍA	Tamaño fuente	El tamaño del texto tiene que ser como mínimo 12 y como máximo 16
3	TIPOGRAFÍA	Cursiva	Validar que no existe texto en cursiva
4	TIPOGRAFÍA	Negrita	Validar que no existen 'muchos' textos en negrita
5	TIPOGRAFÍA	Subrayado	Validar que no existen 'muchos' textos en subrayado
6	TIPOGRAFÍA	Sombreado	Validar que no existen textos con sombreado
7	TIPOGRAFÍA	Mayúscula	Validar que no existen más de un texto en mayúsculas
8	TIPOGRAFÍA	Color fuente	Validar color NEGRO
9	DISEÑO	Color fondo diapositiva	Validar color BLANCO sólido
10	DISEÑO	Cantidad palabras	Validar que la cantidad de palabras en la diapositiva no supere el límite establecido

Tabla 1 - Listado de requisitos

En los siguientes aparatados se detallará más en profundidad cada una de las funcionalidades analizándolas mediante la identificación de los requisitos, así como la especificación de casos de uso.

4.2 Identificación de requisitos

En este apartado se detallará de forma ordenada, clara y sencilla cada una de las especificaciones que tiene que abordar el sistema, así como todas sus características.

Estas especificaciones han sido estipuladas mediante reuniones con el tutor del proyecto que ejercía el rol del cliente, además de las aportaciones del ingeniero para complementar las que la tutoría proponía.

El listado de requisitos ha sufrido diversos cambios durante a lo largo de estas reuniones, ya que en algunos casos se determinó suprimir alguna funcionalidad que en un principio el sistema si tendría o también la modificación de algunos de estos requisitos ya que se dio un punto de vista diferente y con él se mejoró su definición. Con estas modificaciones lo que se busca es la mejora y la optimización de la aplicación.

4.2.1 Tabla de requisitos y nomenclatura

Ahora se procede a la explicación de cada uno de los campos de los que consta la tabla de definición de requisitos.

Número de requisito	
Nombre de requisito	
Tipo	☒ Requisito ☐ Restricción
Características:	
Descripción	
Prioridad del requisito	☒ Alta/Esencial ☐Media/Deseado ☐ Baja/Opcional

Tabla 2 - Plantilla requisitos

Numero de requisito: Identificador único por requisito este campo está definido de la siguiente forma

RF ID o RNF ID siendo:

- RF: Requisito funcional
- RNF: Requisito no funcional
- ID: Identificador numérico del requisito

Nombre de requisito: Nombre simple y sencillo del requisito

Tipo: Se distinguen dos posibles valores

- Requisito: Necesidad del sistema
- Restricción: Prohibición de comportamiento

Características: Idea concisa que englobe el requisito

Descripción: Breve explicación del requisito

Prioridad del requisito: indica el grado de importancia del orden de implementación. Pudiendo determinar entre los siguientes niveles:

- Alta/Esencial
- Media/Deseado
- Baja/Opcional

4.2.2 Requisitos funcionales

Requisitos encargados de especificar todas las funcionalidades que el sistema debe satisfacer

Número de requisito	RF01
Nombre de requisito	Subir fichero
Tipo	☒ Requisito ☐ Restricción
Características:	Los usuarios deberán subir al sistema el fichero que quieren validar.
Descripción	El sistema debe permitir y proporcionar una interfaz para la subida del fichero.
Prioridad del requisito	☒ Alta/Esencial ☐Media/Deseado ☐ Baja/Opcional

Tabla 3 - Requisito funcional 1

Número de requisito	RF02
Nombre de requisito	Validar formato fichero
Tipo	☒ Requisito ☐ Restricción
Características:	El formato del fichero permitido debe ser únicamente XML.
Descripción	El sistema debe validar el formato del fichero subido por el usuario. Este formato debe ser XML, de no ser asi el sistema deberá lanzar un mensaje de error.
Prioridad del requisito	☒ Alta/Esencial ☐Media/Deseado ☐ Baja/Opcional

Tabla 4 - Requisito funcional 2

Número de requisito	RF03
Nombre de requisito	Validar estándares
Tipo	☒ Requisito ☐ Restricción
Características:	El fichero subido por el usuario será analizado por el sistema en cuanto a los estándares de Metodología de Lectura Fácil.
Descripción	El sistema validará el fichero subido por el usuario validando cada uno de los estándares fijados, devolviendo T o F y almacenará el resultado.
Prioridad del requisito	☒ Alta/Esencial ☐Media/Deseado ☐ Baja/Opcional

Tabla 5 - Requisito funcional 3

Número de requisito	RF04
Nombre de requisito	Calcular puntuación.
Tipo	☒ Requisito ☐ Restricción
Características:	El fichero subido por el usuario será analizado por el sistema en cuanto a los estándares de Metodología de Lectura Fácil.
Descripción	El sistema recuperará los resultados asociados al fichero subido por el usuario y realizará el recuento tantos de los T como de los F y en función de esto devolverá la puntuación.
Prioridad del requisito	☒ Alta/Esencial ☐Media/Deseado ☐ Baja/Opcional

Tabla 6 - Requisito funcional 4

Número de requisito	RF05
Nombre de requisito	Proporcionar feedback
Tipo	☒ Requisito ☐ Restricción
Características:	El fichero subido por el usuario será analizado por el sistema en cuanto a los estándares de Metodología de Lectura Fácil.
Descripción	El sistema proporcionará por cada estándar con valor devuelto F información sobre porque se ha dado ese resultado y como se podría mejorar.
Prioridad del requisito	☒ Alta/Esencial ☐Media/Deseado ☐ Baja/Opcional

Tabla 7 - Requisito funcional 5

Número de requisito	RF06
Nombre de requisito	Exportar Resultados
Tipo	☒ Requisito ☐ Restricción
Características:	El fichero subido por el usuario será analizado por el sistema en cuanto a los estándares de Metodología de Lectura Fácil.
Descripción	El sistema proporcionaría la opción de exportar a CSV los resultados.
Prioridad del requisito	☐ Alta/Esencial ☒ Media/Deseado ☐ Baja/Opcional

Tabla 8 - Requisito funcional 6

Número de requisito	RF07
Nombre de requisito	Enviar email
Tipo	☒ Requisito ☐ Restricción
Características:	Envío de resultados por email.
Descripción	El sistema debe disponer de la opción de enviar por email los resultados obtenidos al usuario.
Prioridad del requisito	☐ Alta/Esencial ☒ Media/Deseado ☐ Baja/Opcional

Tabla 9 - Requisito funcional 7

4.2.3 Requisitos no funcionales

Estos requisitos representan atributos de calidad de los que el sistema debe de estar dotado.

Atributos como

- Disponibilidad
- Accesibilidad
- Usabilidad
- Estabilidad
- Portabilidad
- Costo
- Operatividad
- Interoperabilidad
- Escalabilidad
- Concurrencia
- Mantenibilidad
- Interfaz
- Seguridad

Número de requisito	RNF01
Nombre de requisito	Accesibilidad
Tipo	☒ Requisito ☐ Restricción
Características:	Herramienta accesible vía web
Descripción	La herramienta debe ser una aplicación web, en la que el usuario podrá acceder a través del acceso a un servidor web mediante la utilización de un navegador.
Prioridad del requisito	☒ Alta/Esencial ☐Media/Deseado ☐ Baja/Opcional

Tabla 10 - Requisito no funcional 1

Número de requisito	RNF02
Nombre de requisito	Interfaz
Tipo	☒ Requisito ☐ Restricción
Características:	Interfaz sencilla, simple e intuitiva.
Descripción	La herramienta presentara una interfaz de usuario sencilla, simple intuitiva y cuidando los detalles para que la experiencia del usuario sea plenamente satisfactoria.
Prioridad del requisito	☒ Alta/Esencial ☐Media/Deseado ☐ Baja/Opcional

Tabla 11 - Requisito no funcional 2

Número de requisito	RNF03
Nombre de requisito	Errores
Tipo	☒ Requisito ☐ Restricción
Características:	Control de errores
Descripción	La herramienta proporcionara en todo momento feedback al usuario de los errores o infracciones cometidas mediante mensajes de aviso.
Prioridad del requisito	☒ Alta/Esencial ☐Media/Deseado ☐ Baja/Opcional

Tabla 12 - Requisito no funcional 3

Número de requisito	RNF04
Nombre de requisito	Precargar Datos
Tipo	☒ Requisito ☐ Restricción
Características:	Previa carga de datos
Descripción	La herramienta tendrá precargada toda la información referente a los estándares.
Prioridad del requisito	☒ Alta/Esencial ☐Media/Deseado ☐ Baja/Opcional

Tabla 13 - Requisito no funcional 4

Número de requisito	RF05
Nombre de requisito	Ayuda
Tipo	☒ Requisito ☐ Restricción
Características:	Manual de usuario
Descripción	El sistema deberá disponer de un manual de usuario para facilitar la navegación y uso de la herramienta
Prioridad del requisito	☐ Alta/Esencial ☒Media/Deseado ☐ Baja/Opcional

Tabla 14 - Requisito no funcional 5

Número de requisito	RNF06
Nombre de requisito	Exportación
Tipo	☒ Requisito ☐ Restricción
Características:	Formato de exportación
Descripción	El fichero CSV que generará la herramienta seguirá un formato específico y prefijado. En el mismo queda definida toda la información relacionada los resultados obtenidos en la validación de la diapositiva. Los campos del fichero son: ● ID Resultado: Identificador del resultado ● Estándar tipo : Tipo asociado al estándar validado ● Estándar descripción: Breve descripción del estándar ● Código Error: Valor dicotómico 0 - 1 ● Mensaje error: Descripción del error provocado, este campo será vacío en caso de que el código de error sea 0.
Prioridad del requisito	☐ Alta/Esencial ☒Media/Deseado ☐ Baja/Opcional

Tabla 15 - Requisito no funcional 6

4.3 Casos de uso

A continuación, se muestran diferentes casos de usos. Este ejercicio tiene como finalidad una mayor compresión de la lógica que ha de implementar el sistema. En ellos se detallarán cada una de las diferentes interacciones entre un usuario y el sistema.

Este sistema nos ayuda también a ver de forma más clara cómo será la relación del usuario con el sistema y las acciones que pueden realizar en él. Con este ejercicio también validaremos la especificación de requisitos, es decir si cumplen totalmente la funcionalidad que el usuario demanda del sistema.

Estos casos de uso están representados con el lenguaje UML a continuación, se detalla una breve descripción sobre los elementos utilizados para esta escenificación de los casos de uso.

El diagrama de casos de uso representa la forma en como un Cliente (Actor) opera con el sistema en desarrollo, además de la forma, tipo y orden en como los elementos interactúan (operaciones o casos de uso).

Un diagrama de casos de uso consta de los siguientes elementos:

- **Actor.**
- **Casos de Uso.**
- **Relaciones de Uso, Herencia y Comunicación.**

Elementos

- **Actor:**

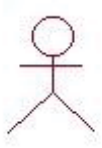


Ilustración 13 - Representación actor

Una definición previa, es que un Actor es un rol que un usuario juega con respecto al sistema. Es importante destacar el uso de la palabra rol, pues con esto se especifica que un Actor no necesariamente representa a una persona en particular, sino más bien la labor que realiza frente al sistema.

- **Caso de Uso:**

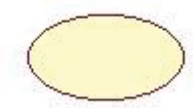
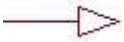


Ilustración 14 - Representación caso de uso

Es una operación/tarea específica que se realiza tras una orden de algún agente externo, sea desde una petición de un actor o bien desde la invocación desde otro caso de uso.

- **Relaciones:**
  - Asociación 
  - Es el tipo de relación más básica que indica la invocación desde un actor o caso de uso a otra operación (caso de uso). Dicha relación se denota con una flecha simple.
  - Dependencia o Instanciación 
  - Es una forma muy particular de relación entre clases, en la cual una clase depende de otra, es decir, se instancia (se crea). Dicha relación se denota con una flecha punteada.
  - Generalización 
  - Este tipo de relación es uno de los más utilizados, cumple una doble función dependiendo de su estereotipo, que puede ser de Uso (<<uses>>) o de Herencia (<<extends>>).

- **Escenario:** establece los límites del sistema. En el diagrama de casos de uso, el escenario queda representado de la siguiente forma:

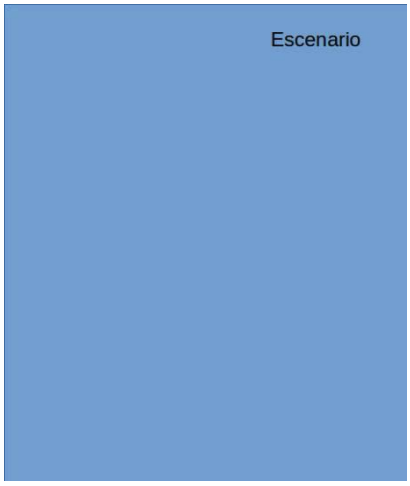


Ilustración 15 - Representación escenario

(Referencias [26] de la bibliografía)

Como una primera aproximación identificamos al actor que interactuara con el sistema:

- Usuario de la aplicación de validación de diapositiva: representa al usuario que interactúe con la aplicación de validación de diapositiva, pudiendo acceder a todas las funcionalidades del sistema de validación de diapositiva.

Cada caso de uso o diagrama de uso ira acompañado con una tabla resumen en la que se especificaran de forma formal y detallada cada uno de los casos. En esta tabla además se recogerán las condiciones que se han de dar para que el caso de uso pueda ser ejecutado, así como el rol que es capaz de desarrollarlo.

A continuación, se muestra una plantilla de la tabla comentada:

Identificador	
Nombre	
Objetivo	
Actor	
Precondiciones	
Postcondiciones	
Escenario	

Tabla 16 - Plantilla caso de prueba

- **Identificador**: CU E0 – 00 Siendo CU (Caso de Uso), E0 (Escenario 0) y 00 el número de identificación del caso de uso.
- **Objetivo**: Breve descripción del objetivo.
- **Actor**: Actor que interacciona con el caso de uso en cuestión.
- **Precondiciones**: Condiciones que deben cumplirse de forma previa para realizar la operación.
- **Postcondiciones**: Estado del sistema después de la realización de la operación.
- **Escenario básico:** Límites del sistema, descripción por fases para el cumplimiento del caso de uso.

A continuación, se detallan cada uno de los casos de uso establecidos para el sistema.

Identificador	**CU E01**
Nombre	Navegación
Objetivo	Navegación del usuario por la página web del sistema
Actor	Usuario de la aplicación de validación de diapositiva
Precondiciones	-
Postcondiciones	-
Escenario	1. Se accede a la aplicación 2. Inserción de fichero de diapositiva 3. Visualización de resultados 4. Exportación resultados

Tabla 17 - Caso de uso 1

Identificador	**CU E02**
Nombre	Inserción del fichero
Objetivo	Insertar el fichero a validar
Actor	Usuario de la aplicación de validación de diapositiva
Precondiciones	Estar dentro del sistema
Postcondiciones	-
Escenario	1. Se accede a la aplicación 2. Se insertar el correspondiente fichero en el input

Tabla 18 - Caso de uso 2

Identificador	**CU E03**
Nombre	Visualización de resultados
Objetivo	Visualizar los resultados obtenidos tras la validación del sistema de la diapositiva
Actor	Usuario de la aplicación de validación de diapositiva
Precondiciones	1. Estar dentro del sistema 2. Insertar fichero a validar
Postcondiciones	-
Escenario	1. Se accede a la aplicación 2. Se insertar el correspondiente fichero en el input

Tabla 19 - Caso de uso 3

Identificador	**CU E04**
Nombre	Exportación resultados
Objetivo	Exportar a fichero CSV los resultados obtenidos tras la validación de la diapositiva.
Actor	Usuario de la aplicación de validación de diapositiva
Precondiciones	1. Estar dentro del sistema 2. Insertar fichero 3. Visualizar resultados
Postcondiciones	-
Escenario	1. Se accede a la aplicación 2. Se insertar el correspondiente fichero en el input 3. Se visualizan los resultados 4. Se descarga los resultados

Tabla 20 - Caso de uso 4

Identificador	**CU E05**
Nombre	Salida del sistema
Objetivo	Abandonar la aplicación y las operaciones realizadas en la misma.
Actor	Usuario de la aplicación de validación de diapositiva
Precondiciones	Estar dentro del sistema
Postcondiciones	-
Escenario	En cualquier fase del proceso de creación de escenarios el usuario puede abandonar la aplicación de forma inmediata.

Tabla 21 - Caso de uso 5

En el siguiente diagrama se muestra el flujo general del sistema, el usuario es el único actor que realiza acciones en el escenario. Este escenario básico representa el sistema en su totalidad y abarca todos los posibles casos generales que se pueden dar como son la visualización del contenido de la página, la inserción del fichero a validar, la posterior visualización de los resultados obtenidos, la exportación de los resultados obtenidos y finalmente la salida del sistema.

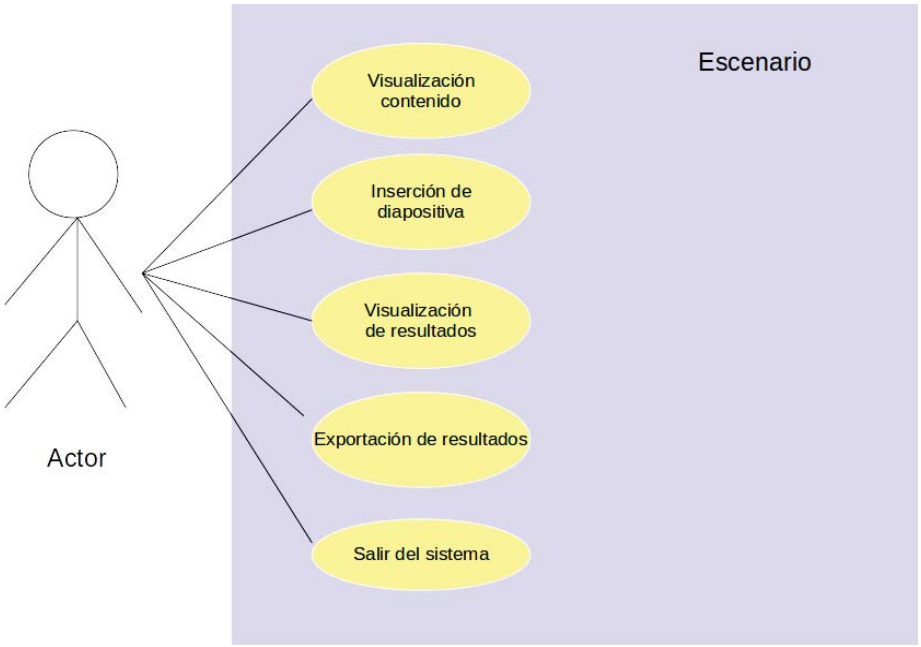


Ilustración 16 - Flujo general aplicación

Casos de uso del sistema

En este apartado se va a detallar el escenario general pero esta vez de la parte interna del sistema sin contar con ningún rol que actué sobre el mismo. Este caso de uso se podría entender como las acciones que realiza el sistema por debajo cuando se ejecuta el diagrama general descrito anteriormente.

El flujo del proceso sería el almacenamiento del fichero subido por el usuario en el sistema, la grabación en base de datos y finalmente la exportación del fichero de resultados.

Identificador	**CU E06**
Nombre	Almacenamiento de fichero
Objetivo	El fichero introducido por el usuario se almacenará dentro del sistema
Actor	-
Precondiciones	-
Postcondiciones	-
Escenario	Siempre que un usuario introduce un fichero, el sistema recoge dicho fichero y lo almacena.

Tabla 22 - Caso de uso 6

Identificador	**CU E07**
Nombre	Almacenado de información de ficheros y resultados en la base de datos.
Objetivo	Almacenar información del fichero introducido y de los resultados devueltos por el sistema
Actor	-
Precondiciones	-
Postcondiciones	-

Escenario	El sistema siempre almacenara en base de datos la información acercar del fichero introducido, así como el registro de los resultados devueltos tras la validación.

Tabla 23 - Caso de uso 7

Identificador	**CU E09**
Nombre	Exportación resultados
Objetivo	Exportar a fichero CSV los resultados obtenidos tras la validación de la diapositiva.
Actor	-
Precondiciones	-
Postcondiciones	-
Escenario	Al final el proceso de validación del fichero el sistema dará la opción de exportar los resultados en formato CSV

Tabla 24 - Caso de uso 8

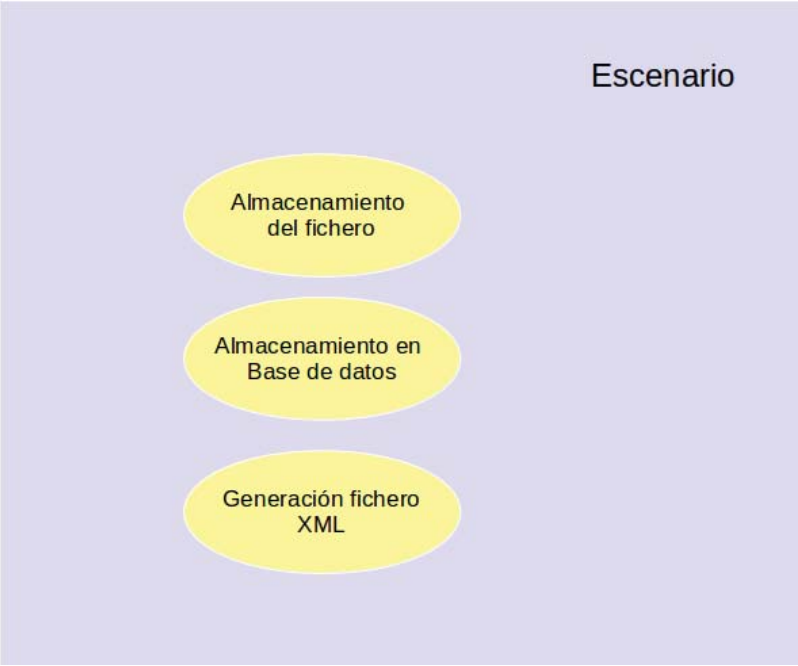


Ilustración 17 – Procesos principales del sistema

5. Solución del problema

5.1 Diseño

En este apartado sobre el diseño del sistema se detallará la definición de la arquitectura del mismo tanto a nivel software como a nivel hardware, así como la especificación de forma detallada de los diferentes componentes y la infraestructura tecnológica necesaria para dar un soporte óptimo para el correcto funcionamiento de la aplicación.

Como se ha comentado en capítulos anteriores, la arquitectura Cliente/Servidor se puede dividir en diferentes capas. En la siguiente imagen se muestra la división de estas capas con los módulos implicados en cada una de ellas y con sus tecnologías correspondientes.

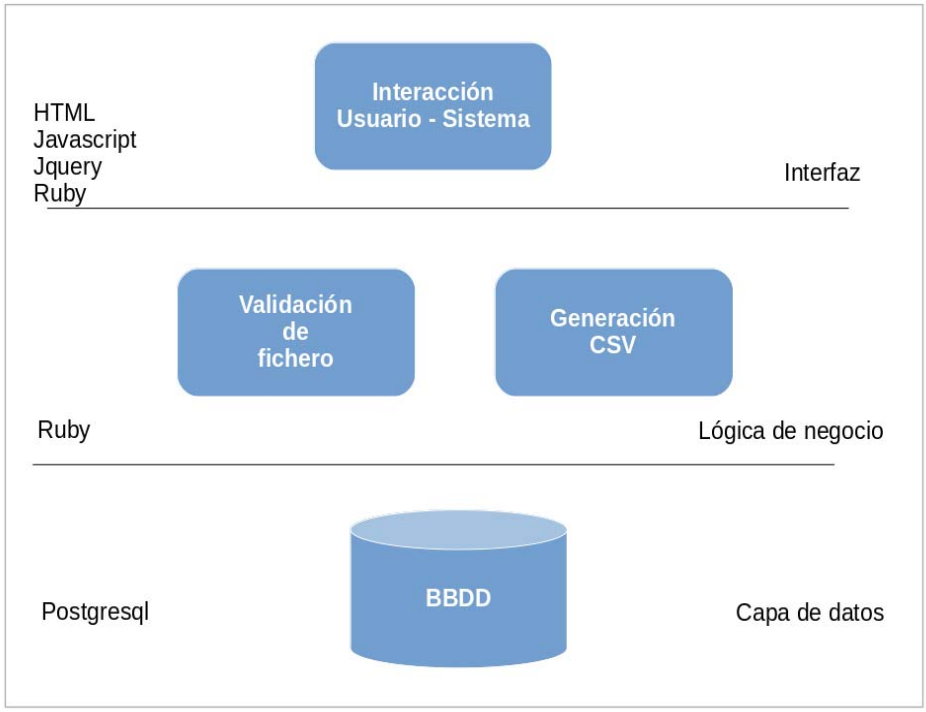


Ilustración 18 - Capas del sistema

En los siguientes puntos se dará una explicación detallada de cada una de las secciones expuestas en la imagen anterior.

5.1.1 Lógica de negocio

Con respecto a la lógica de negocio, esta capa está compuesta por dos módulos:

1. Validación del fichero
2. Exportación a CSV

Estos módulos se explicarán con detalle en secciones posteriores.

La lógica de negocio es la encargada de realizar el control de todas y cada una de las fases de proceso de validación. Desde este módulo se accede a base de datos para grabar información sobre el fichero introducido por el usuario y posteriormente para realizar consultas de recuperación de información necesaria para llevar a cabo la validación del fichero y la visualización de los resultados. En este módulo es donde se encuentra toda la lógica de la aplicación toda la funcionalidad del sistema en donde se controlan los múltiples aspectos del mismo.
En esta capa se encuentra la conexión con la base de datos y la lógica implementada tanto en la validación del fichero como en la generación del fichero CSV de resultados.

Este módulo tiene el papel del controlador ya que es donde se encuentran todos aquellos componentes que son los encargados de controlar y establecer el comportamiento de las diferentes fases del proceso, así como el control del correcto funcionamiento de las mismas.

A continuación, se detallará más en profundidad los módulos que conciernen esta capa de lógica de negocio.

Validación de fichero

En este módulo se controla todo el funcionamiento relaciona con la validación del fichero de la diapositiva con el fin de otorgar una puntuación al fichero al finalizar la misma.

Este módulo coordina no sólo la comprobación de los estándares en el fichero (diapositiva) si no también controla el alta de cada uno de los resultados respecto a cada estándar que se valida, así como el alta del fichero introducido. También es el encargado de recuperar la información necesaria sobre los estándares para poder ser validados en el fichero introducido.

La funcionalidad de la validación en sí está implementada mediante un flujo de comprobaciones sobre el fichero introducido, comprobaciones acerca de cada uno de los estándares. Por ello en este momento es como se a comentado anteriormente cuando es necesario recuperar de la capa de datos la información sobre los estándares para poder proceder a la validación de los mismos sobre el fichero.

En función de los estándares que el fichero cumpla o no se formará una vista en la que se mostrará un listado con todos los estándares diferenciando los que han sido pasados de forma satisfactoria y los que no.

Generación del CSV

Este es el módulo encargado de realizar la generación del documento CSV con los resultados devueltos por el proceso de validación del fichero. En él se vuelcan todos los resultados obtenidos para ello este módulo al igual que el anterior se apoya en la capa de datos para poder recuperar de base de datos los resultados almacenados por el módulo anterior.

5.1.2 Capa de datos

Esta capa está compuesta por las tablas creadas en la base de datos. Dichas tablas contienen la información de los ficheros que son cargados en la aplicación, la información acerca de los estándares y finalmente información de los resultados obtenidos. Desde la lógica de negocios se recupera toda esta información para el correcto funcionamiento de la aplicación y por medio de accesos entre ambas capas se obtienen datos que serán determinantes para el comportamiento de la herramienta en diversas fases del proceso de validación.

A continuación, se presenta la definición del diseño de la Base de Datos de esta aplicación, mostrando el diagrama Entidad-Relación de la misma.

Diagrama ENTIDAD-RELACIÓN

Es una herramienta para el modelado de datos que permite representar las entidades relevantes de un sistema de información así como sus interrelaciones y propiedades.

Entidad. Son las tablas en sí misma los objeto ‘clases’ que permitirán almacenar las diferentes instancias que se generen de ellos.

Atributos - Son cada una de las propiedades que definen al objeto que pretendemos almacenar.

Relación. Es un vínculo que definen la dependencia de dos o más entidades. Esto es definir la dependencia de unos objetos con otros es decir de unas tablas con otras.

Clave. Es el campo o atributo de una entidad o tabla que tiene como objetivo distinguir cada registro del conjunto, sirviendo sus valores como datos vinculantes de una relación entre registros de varias tablas.

Integridad referencial. Se denomina integridad referencial al tipo de interrelación que se produce entre tablas mediante un campo clave que deberá contener la cadena alfanumérica exacta al identificador de la tabla auxiliar para poder realizar la relación entre los registros. En caso contrario no se produce la relación. Además, se trata de un mecanismo que evita duplicidades e incorrecciones ya que la propiedad de integridad referencial conmina a que los datos de un usuario además de su identificador ID sean distintos al de los demás. Dicho de otra forma, no pueden existir dos registros iguales con los mismos datos.

Tipos de relaciones

- Según cardinalidad. La cardinalidad se representa en un diagrama ER como una etiqueta que se ubica en ambos extremos de la línea de relación de las entidades y que puede contener diversos valores entre los que destacan comúnmente el 1 y el *, obteniendo los siguientes tipos:

    - Relación 1 a 1. La relación uno a uno, define que un único registro de la tabla puede estar relacionado con un único registro de la tabla relacionada.

    - Relación 1 a *. La relación de uno a varios define que un registro dado de una tabla auxiliar o secundaria sólo puede estar vinculado con un único registro de la tabla principal con la que está relacionada.

    - Relación * a *. La relación de varios a varios define que un registro de una tabla puede estar relacionado con varios registros de la tabla relacionada y viceversa.

(Referencia [27] de la bibliografía)

A continuación, se muestran y se detallan modelos conceptual, lógico y físico de la base de datos del sistema.

Modelo conceptual

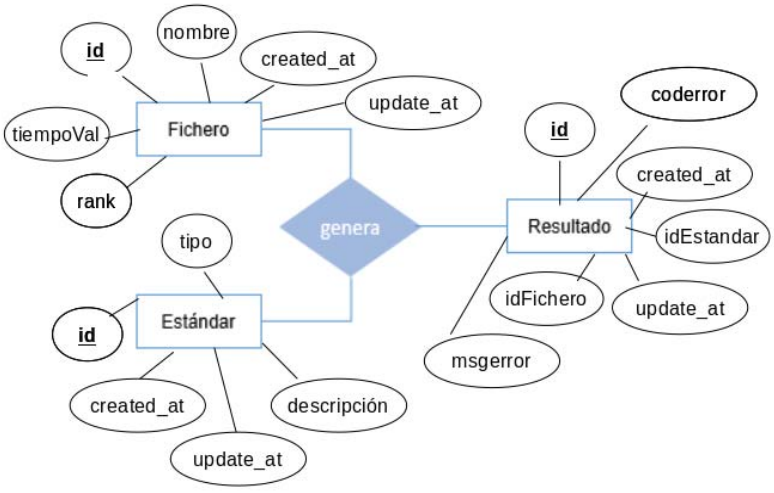


Ilustración 19 - Modelo conceptual

En este diagrama se pueden observar las diferentes entidades de las que está compuesto el sistema.

- Estándar
- Fichero
- Resultado

La entidad **Estándar** posee los siguientes atributos:

1. id: Identificador único del fichero
2. tipo: Especifica la categoría a la que pertenece el estándar.
3. descripción: Breve especificación del estándar
4. created_at: Fecha de creación del registro
5. updated_at: Fecha de la última modificación del registro

Por su parte la entidad **Fichero** posee los siguientes atributos:

1. id: Identificador único del fichero.
2. nombre: nombre del archivo
3. tiempoVal: tiempo dedicado a su evaluación
4. rank: Puntuación obtenida tras la validación
5. created_at: Fecha de creación del registro.
6. updated_at Fecha de la última modificación del registro.

Finalmente se tiene la entidad **Resultado** con los siguientes atributos:

1. id: Identificador único del resultado
2. idFichero: Identificador del fichero
3. id Estándar: Identificador del estándar.
4. coderror: Codigo de error.
5. msgerror: Mensaje de error.

Con respecto a las relaciones las tablas Estándar y Fichero convergen en la tabla Resultado a través de la relación ‘genera’. Realmente la tabla Resultado es el resultado de la relación N:M producida por las otras entidades. Esto es así porque En un fichero se validaron todos los estándares y un estándar será validado en uno o más fichero. Por lo tanto, aquí se tiene la relación muchos a muchos. El tipo de relaciones N:M por norma general tabla, por lo que la entienda Resultado surge de esta relación y de ahí que esta entidad posea las claves principales de las entidades Fichero y Estándar.

Modelo lógico

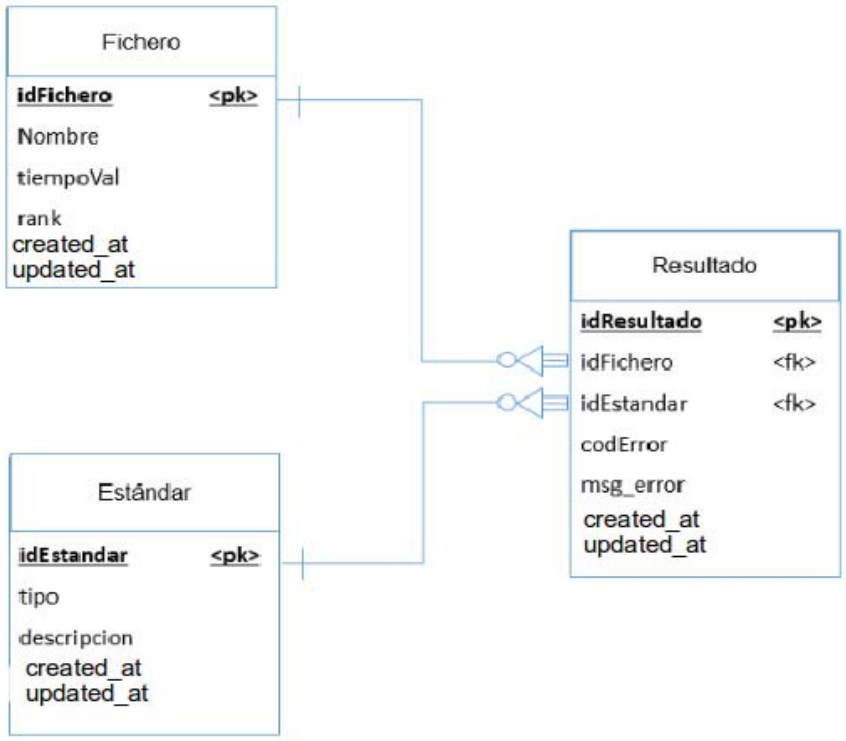


Ilustración 20 - Modelo lógico

En este modelo podemos ver de forma más clara cómo están relacionadas las entidades.
Se observa como la entidad Resultado recoge las claves principales de las entidades Fichero y Estándar.

Se ha de mencionar que, aunque la entidad Resultado es una entienda que surge de la relación N:M como se ha comentado anteriormente esta entidad Resultado tiene su propia clave principal esto es así porque el lenguaje de programación utilizado para implementar este sistema, Ruby on Rails (en el que más adelante se profundizará) por defecto pone un id a todas las tablas o mejor dicho modelos que se generan.

Modelo físico

En este modelo podemos ver el desglose de cada uno de los atributos de las diferentes entidades con el tipo de dato asignado a cada uno de ellos y con un ejemplo de los mismos.

Entidad Fichero

CAMPO	CLAVE	OBLIGATORIO	TIPO DE DATO	EJEMPLO
idFichero	PK	SI	integer	1
nombre			string	presentacion.xml
tiempoVal			double	12,30
rank			integer	5
created_at		SI	datetime	2016-03-03 20:31:27.637536
update_at			datetime	2016-03-03 20:31:27.637536

Tabla 25 - Entidad Fichero

Entidad Estándar

CAMPO	CLAVE	OBLIGATORIO	TIPO DE DATO	EJEMPLO
idEstandar	PK	SI	integer	1
tipo		SI	string	TIPOGRAFIA
descripcion		SI	string	Validar que no existe texto en cursiva
created_at		SI	datetime	2016-03-03 20:31:27.637536
update_at			datetime	2016-03-03 20:31:27.637536

Tabla 26 - Entidad Estándar

Entidad Resultado

CAMPO	CLAVE	OBLIGATORIO	TIPO DE DATO	EJEMPLO
idResultado	PK	SI	integer	1
idFichero	FK	SI	integer	1
idEstandar	FK	SI	integer	1
codError		SI	integer	2
msg_error			string	Formato de texto inválido: no se admiten cursivas
created_at		SI	datetime	2016-03-03 20:31:27.637536
update_at			datetime	2016-03-03 20:31:27.637536

Tabla 27 - Entidad Resultado

PASO A TABLAS DE CADA ENTIDAD

En este apartado se puede observar ejemplos de los registros almacenados en cada una de las entidades que conforman el sistema.

Entidad Fichero

idFichero	nombre	tiempoVal	rank	created_at	update_at
1	pre01.xml	45	5	2016-03-03 20:31:27.637536	2016-03-03 20:31:27.637536
2	biodiesel.xml	2,4	3	2015-01-03 13:17:40.987302	2015-01-03 13:17:40.987302
3	leds.xml	1,4	2	2017-01-02 14:20:01.789325	2017-01-02 14:20:01.789325

Tabla 28 - Tabla Fichero

Entidad Estándar

idEstandar	tipo	descripcion	created_at	update_at
1	TIPOGRAFIA	El tamaño del texto tiene que ser como mínimo 12 y como máximo 16	2017-01-02 14:20:01.789325	2016-03-03 20:31:27.637536
2	TIPOGRAFIA	Tipo de fuente del texto pertenece a los estilos aceptados	2017-01-02 14:20:01.789325	2015-01-03 13:17:40.987302
3	TIPOGRAFIA	No existe texto en cursiva	2017-01-02 14:20:01.789325	2017-01-02 14:20:01.789325

Tabla 29 - Tabla Estándar

Entidad Resultado

idResultado	idFichero	idEstandar	codError	msg_error	created_at	update_at
1	1	1	1	Se han detectado tamaños de fuente no válidos	2017-03-02 17:05:03.651039	2016-03-03 20:31:27.637536
2	2	3	1	Formato de texto inválido: no se admiten cursivas	2017-01-02 14:20:01.789325	2015-01-03 13:17:40.987302
3	2	2	0		2017-01-02 14:20:01.968753	2017-01-02 14:20:01.789325

Tabla 30 - Tabla Resultado

5.1.3 Interfaz, capa de presentación

La interfaz permite a los usuarios interactuar con la aplicación Web. A través de los componentes de la interfaz se renderizan y se formatea la información que se quiere presentar, así como capturan y validan las entradas de datos. Ejemplos clásicos son los botones, las etiquetas, los hipervínculos, los cuadros de texto, las tablas, etc.
La mayoría de estos componentes vienen incluidos dentro del propio lenguaje HTML y otros se implementan en lenguajes de scripting para el navegador como Javascript.

Ilustración 21 - Capa de presentación

Definición del aspecto de la aplicación

En este apartado se muestran las diferentes pantallas o vistas que componen la aplicación:

Vista inicial

Ilustración 22 - Vista inicial

En la vista inicial de la aplicación se muestra una breve definición sobre Lectura Fácil, seguido de los objetivos que tiene el sistema, los servicios que presta y finalmente un formulario de contacto.

En la parte superior de la vista tenemos un menú por el que la aplicación permite su navegación por los diferentes apartados citados anteriormente.

Seguido de este menú se puede ver el componente input que permite la inserción del fichero a validar. Debajo justo se encuentra el componente de tipo botón para proceder a la validación del fichero.

Vista resultados

En esta vista se encuentra la puntuación final obtenida tras el proceso de validación del fichero. Se puede observar tanto los estándares que se han si pasado de forma satisfactoria y los que por el contrario no se cumplen en la diapositiva validada.
Seguidamente nos encontramos con el botón de descarga, este botón nos ofrece la posibilidad de exportar todos los resultados en un fichero con formato CSV.

Los estándares que se cumple en la diapositiva se marcan en color verde, pero también con el icono del 'validado' esto está implementado así debido a que la aplicación está pensada para personas que tengan algún tipo de dificultad a la hora de distinguir los colores. Por ello los estándares que no se cumplen en la diapositiva además de estar marcados en color rojo tienen asociado el icono del aspa para una mejor comprensión de los resultados.

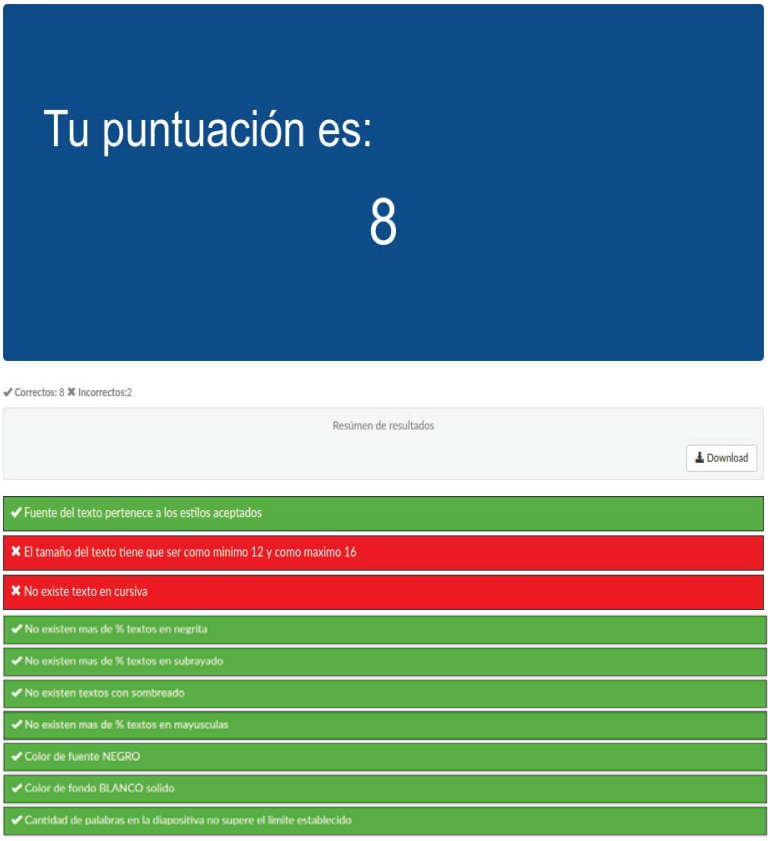


Ilustración 23 - Vista Resultados

Vista resultados desglosados

Los estándares que no se han superado tiene la opción de hacer clic en ellos. Con este clic se despliega un panel en el que se muestra el porqué no se ha superado este estándar. En el panel se muestran los siguientes datos:

- **Fichero**: Recoge el nombre del fichero que ha sido validado.
- **Estándar**: Descripción del estándar.
- **Mensaje de error**: Aporta información sobre el error cometido
- **Ayuda**: Pop up que nos informa sobre qué podemos hacer para satisfacer este estándar.

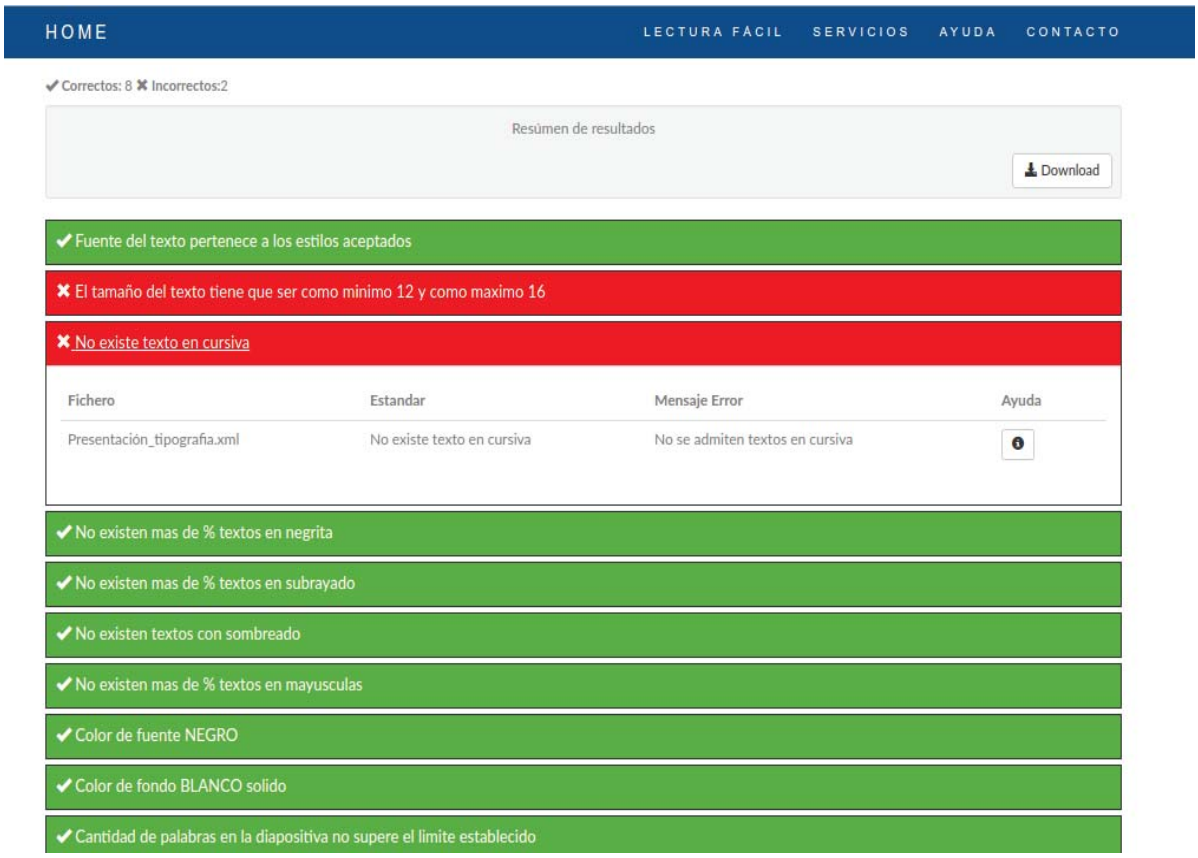


Ilustración 24 - Vista Resultados desglosados

6. Implementación

En este capítulo de la documentación se describe como se ha llevado a cabo la implementación de la arquitectura y el diseño del sistema, que ha sido detallado en los apartados anteriores. Se resaltaron las tecnologías utilizadas en dicha implementación, así como los aspectos para importantes en la fase de desarrollo del sistema.

6.1 Tecnologías y herramientas utilizadas

Como ya se comentó en el apartado del estado del arte, la aplicación está desarrollada en el lenguaje de programación Ruby bajo el framework Rails.

- La parte de front está desarrollada con Ruby, HTML, Javascript, JQuery y Bootstrap.
- Todo el proyecto se ha desarrollado con el editor Atom.
- El control de versión se ha llevado a cabo mediante la herramienta Github.
- La publicación de la aplicación en 'producción' se ha llevado a cabo por medio de la plataforma Heroku.

6.2 Desarrollo del proyecto

Como se comentó en el párrafo anterior la parte de backend de la aplicación está desarrollada en Ruby on Rails, a continuación, se explican y comentan los puntos más importantes de la fase de desarrollo de la aplicación.

6.2.1 Instalación Ruby

La instalación de Ruby es bastante sencilla y simple de hacer. En este caso se trabajó con el sistema operativo Ubuntu por lo que todo lo que se comente sobre instalación acerca tanto de Ruby on Rails como del resto de herramientas está explicada para este sistema operativo.

Pasos para la instalación de Ruby on Rails en Ubuntu

Como primer paso se ha de mencionar que la instalación de Ruby y el framework Rails se instalan de forma independiente. A continuación, se muestran los comandos necesarios para la instalación de Ruby en un equipo Ubuntu.

Usan el gestor de paquetes apt. la instalación se realiza de la siguiente forma.

```
$ sudo apt-get install ruby-full
```

El siguiente paso es verificar que ruby se instaló correctamente en el equipo. Esto se puede verificar ejecutando el siguiente comando desde el terminal.

```
ruby -v
```

Es importante destacar que este proyecto se ha desarrollado con la versión 2.3.0 de Ruby, por ello es necesario ejecutar el siguiente comando. Este comando nos instalara la versión 2.3.0 de Ruby.

```
rvm install 2.3.0
```

Una vez que ruby se ha instalado con éxito en el equipo se procede a la instalación de Rails. Esto se realiza ejecutando el siguiente comando.

```
gem install Rails
```

Para finalizar con la instalación de Rails se ha de comprobar que la instalación tuvo éxito para ello basta con ejecutar el siguiente comando.

Si la respuesta que se obtienen es algo como Rails 4.0.1, Rails se instaló correctamente.

En el siguiente apartado se explica cómo se genera un nuevo proyecto de Ruby on Rails.

6.2.2 Creación proyecto Rails

Crear un nuevo proyecto de Ruby on Rails es incluso más fácil que la instalación.
Para crear un nuevo proyecto de Ruby on Rails basta con posicionarse desde el terminal de Ubuntu en el directorio en el que se quiere almacenar este proyecto, una vez posicionado en el directorio basta con ejecutar el siguiente comando y el nuevo proyecto de Rails estará creado.

```
Rails new w.easy
```

Estructura del proyecto

A continuación, se pasa a explicar la estructura de un proyecto de Ruby on Rails. Esta estructura consta de una serie de carpetas y archivos que nos ayudan a trabajar de forma ordenada y eficiente.

```
tree -L 2
.
├── Gemfile
├── Gemfile.lock
├── README.md
├── README.rdoc
├── Rakefile
├── app
│   ├── assets
│   ├── controllers
│   ├── helpers
│   ├── mailers
│   ├── models
│   ├── serializers
│   └── views
├── bin
│   ├── bundle
│   ├── Rails
│   └── rake
├── config
│   ├── application.rb
│   ├── boot.rb
│   ├── database.yml
│   ├── environment.rb
│   ├── environments
│   ├── initializers
│   ├── locales
│   └── routes.rb
├── config.ru
├── db
│   ├── migrate
│   ├── schema.rb
│   └── seeds.rb
├── lib
│   ├── assets
│   └── tasks
├── log
│   └── development.log
├── public
│   ├── 404.html
│   ├── 422.html
│   ├── 500.html
│   ├── codehero.png
│   ├── favicon.ico
│   └── robots.txt
├── test
│   ├── controllers
│   ├── fixtures
│   ├── helpers
│   ├── integration
│   ├── mailers
│   ├── models
│   └── test_helper.rb
└── vendor
    └── assets
```

```
29 directories, 24 files
```

A continuación, se destacan los ficheros más relevantes de un proyecto de Ruby on Rails:

- **Gemfile**: En este fichero se almacenan las gemas requeridas por la aplicación. Una gema no es más que una librería que dota a nuestro proyecto funcionalidades adicionales. Este sería un ejemplo de una línea del Gemfile: gem 'nokogiri'
- **Rakefile**: Este fichero almacena las tareas relacionadas con el comando rake. En este proyecto no se ha utilizado este fichero ya que no ha sido necesario programar ninguna tarea.
- **routes.rb:** EN este fichero se configuran las rutas del sistema. Ejemplo: get 'products/:id' => 'catalog#view'
- **database.yml:** En el se guardan los datos de acceso a la base de datos. Rails te permite configurar tres ambientes de base de datos diferentes: desarrollador, pruebas y producción.

Base de datos

Para configurar la base de datos basta con adaptar el archivo *database.yml* para que se conecte con el manejador de base de datos que estemos usando, en este caso postgresql.

Por defecto Rails trae incorporada la conexión con la base de datos sqlite3. Para poder utilizar el manejador de Postgresql es necesario instalarlo previamente mediante el siguiente comando

```
sudo apt-get install postgresql postgresql-contrib libpq-dev
```

Una vez ejecutado el anterior comando es necesario añadir al archivo Gemfile gema de Postgresql:

```
gem install pg
```

Ahora solo faltaría modificar el archivo *database.yml* adaptándolo a la conexión de base de datos que se quiera.

```
development:
 adapter: postgresql
 encoding: utf8
 database: w_easy.development
 username: root
 password: root
 host: localhost
```

```
 port: 3306

test:
 adapter: postgresql
 encoding: utf8
 database: w_easy
 username: root
 password: root
 host: localhost
 port: 3306

production:
 adapter: postgresql
 encoding: utf8
 database: dcr8ri7lg8odgl
 username: --
 password: --
 host: ec2-46-137-97-169.eu-west-1.compute.amazonaws.com
 port: 5432
```

Como se puede observar la conexión de producción está alojada en un servidor de Amazon, esto es así porque producción está alojada en Heroku. Por temas de seguridad no se exponen los datos referentes al usuario y contraseña

Creación de modelos

Los modelos en Rails al igual que en los otros framework MVC son aquellos en donde se mantiene toda la lógica de negocios de nuestra aplicación web. Los modelos desarrollan la información importante de la aplicación móvil, como los accesos a datos, validaciones o clases y métodos de importancia para el sistema.

Es importante mencionar que los modelos de Rails utiliza un nombre singular, y sus correspondientes tablas de la base utilizan un nombre plural.

Para crear un modelo en Rails al igual que los controladores, el framework nos proporciona una línea de comando que hace fácil el trabajo.

```
Rails generate model fichero nombre:string
tiempoval:decimal  rank:integer
```

El resultado que debe aparecer tras la ejecución del comando es:

```
invoke  active_record
     create    db/migrate/20130707222645_create_ficheros.rb
     create    app/models/fichero.rb
```

```
      invoke    test_unit
      create      test/models/ficheros_test.rb
      create      test/fixtures/ficheros.yml
```

Al ejecutar la línea de comando le estamos pidiendo a Rails que genere el modelo necesario para manejar una tabla en base de datos llamada 'fichero' con tres campos (nombre, tiempoval y rank). Estos atributos se agregan automáticamente a la tabla de usuario en la base de datos y se asigna al modelo correspondiente. A parte de estos atributos que se han añadido, Rails añade por defecto tres atributos más: id, created_on y update_on

Como pueden observar se crea un archivo dentro del directorio db/migrate/, este archivo es la declaración de la tabla de donde se basa Rails para mapearla.

Esto es una de las características más importantes y útiles que proporciona Rails, el sistema de migraciones. Cada vez que se crea un nuevo modelo con el comando explicado anteriormente, se genera un archivo de migración. Si por el contrario lo que hacemos es eliminar un modelo Rails también genera un archivo de migración, por lo que tenemos el histórico de migraciones que sufre la base de datos. Esto es muy útil porque Rails permite la migración de la base de datos en cualquier gestor de base de datos. Es decir, es como si Rails genera archivos SQL 'genéricos' que son entendidos por cualquier gestor de base de datos.

A continuación, se muestran algunos ejemplos de estos archivos de migración:

Creacion de la tabla Estandars archivo (20170315174020_create_estandars.rb)

```
class CreateEstandars < ActiveRecord::Migration
 def change
   create_table :estandars do |t|
     t.string :tipo
     t.string :descripcion
     t.datetime :fecha_crea
     t.timestamps null: false
   end
end
end
```

Eliminación de columna

```
class RemoveFechaCrea < ActiveRecord::Migration
 def change
   remove_column :ficheros, :fecha_crea
   remove_column :estandars, :fecha_crea
```

```
   remove_column :resultados, :fecha_crea
 end
end
```

Como se puede ver en el código anterior, para eliminar una columna basta con poner remove_column : nombre de la tabla, :columna a eliminar.

Para correr estos archivos de grabación y que nuestra base de datos recupere estos nuevos cambios y los inyecte en la base de datos del sistema, únicamente hay que ejecutar el siguiente comando:

```
rake db:migrate
```

Creación de controladores

Los Controladores en Rails tienen como propósito mantener separada la lógica de negocios de la aplicación (modelos) de las vistas. Los controladores son los que reciben peticiones, las procesan y muestran la información en un formato legible para los usuarios en las vistas.

Para dar de alta un nuevo controlador basta con ejecutar el siguiente comando:

```
Rails generate controller fichero index
```

Con esta línea estamos pidiendo al framework crear un controlador llamado 'fichero' y el método índex de éste. Cabe destacar que luego pueden incluirse los métodos que se necesiten agregándoles directamente al archivo del controlador y creando vistas a mano sin necesidad de utilizar una línea de comando. A continuación, pueden ver el resultado de la línea de comando:

```
create  app/controllers/fichero_controller.rb
      route  get "fichero/index"
     invoke  erb
     create    app/views/fichero
     create    app/views/fichero/index.html.erb
     invoke  test_unit
     create    test/controllers/fichero_controller_test.rb
     invoke  helper
     create    app/helpers/fichero_helper.rb
     invoke    test_unit
     create      test/helpers/fichero_helper_test.rb
     invoke  assets
     invoke    coffee
     create      app/assets/javascripts/fichero.js.coffee
     invoke    scss
     create      app/assets/stylesheets/fichero.css.scss
```

Lo más importante de todo lo que crea esta línea de comando en el terminal es sin duda el archivo app/controllers/fichero_controller.rb que es el controlador en sí y las vistas asociadas a éste, en este caso es una sola app/views/fichero/index.html.erb.

Creación de scafold

Si bien es cierto que para la creación tanto de los modelos como de los controladores de esta aplicación no se utilizaron los comando descritos anteriormente, para la creación se utilizó un comando que engloba la creación tanto del modelo, controlador y las vistas iniciales.
Este comando es el siguiente:

```
Rails generate scaffold Fichero nombre:string tiempoval:decimal
ran:integer
```

Tras la ejecución de este comando la respuesta devuelta sería:

```
create    db/migrate/20141121012013_create_ficheros.rb
create    app/models/fichero.rb
invoke  resource_route
route    resources :ficheros
invoke  scaffold_controller
create    app/controllers/ficheros_controller.rb
invoke    erb
create      app/views/ficheros
create      app/views/ficheros/index.html.erb
create      app/views/ficheros/edit.html.erb
create      app/views/ficheros/show.html.erb
create      app/views/ficheros/new.html.erb
create      app/views/ficheros/_form.html.erb
invoke    helper
create      app/helpers/ficheros_helper.rb
invoke    jbuilder
create      app/views/ficheros/index.json.jbuilder
create      app/views/ficheros/show.json.jbuilder
invoke  assets
invoke    coffee
create      app/assets/javascripts/ficheros.js.coffee
invoke    scss
create      app/assets/stylesheets/ficheros.css.scss
invoke  scss
create    app/assets/stylesheets/scaffolds.css.scss
```

Como se puede observar en la salida, este comando genera el modelo, el controlador, las diferentes vistas por defecto como son el index, el formulario de edición, creación de nuevo fichero y visualización del fichero creado.

También nos genera automáticamente un helper, que más adelante se hablará sobre el, y nos genera los archivos css y javascript.

Cabe destacar para la creación del scafflod Resultado la estructura varía en un parámetro que hay que añadir dada la relación N:M que existe con los modelos Fichero y Estándar.

A continuación, se deja la línea de código necesaria para la creación del scafflod Resultados:

```
Rails generate scaffold Resultado coderror:integer
msg_error:string fichero_id:references estandar_id:references
```

Como se puede observar para añadir la relación tanto de Resultado con Estándar como con Fichero en Rails hay que añadir el tipo references. Esto hace que ese id se asocie directamente con la clave principal del modelo referenciado.

CONTROLADORES

Por defecto cuando se da de alta un nuevo controlador se generan una serie de método por defecto, estos métodos son: index, show, new, edit, create, update, destroy.

A continuación, se pasa a explicar cada uno de estos métodos. Es necesario destacar que los tres controladores iniciales que han sido creados para esta aplicación tienen la misma estructura de métodos.

Método index

Este método devuelve en la variable global @ficheros todos los ficheros registrados en el modelo Fichero, es decir todos los registros almacenados en la tabla Ficheros.

```
def index
   @ficheros = Fichero.all
end
```

Método new

Este método crea un nuevo objeto Fichero y lo asocia a la variable global @fichero

```
  def new
   @fichero = Fichero.new
 end
```

Método create

Este método crea un nuevo objeto Fichero y asigna a los atributos del objeto los parámetros recibidos. Se podría decir que es el ‘constructor’ del modelo.

Una vez instanciado el objeto comprueba si la grabación tiene éxito o no es decir cuando realiza el grabado en base de datos del objeto si el commit se realizó correctamente o no.

Si el commit tuvo éxito se mostrará un mensaje flash avisando de que el fichero fue creado correctamente. Si por el contrario hubo algún problema al realizar el grabado se mostrará un mensaje de error.

Los fallos al hacer el commit pueden ser por diversas razones, una de las más comunes es que antes incluso de llegar a hacer el commit, el registro sea rechazado porque no cumple con las validaciones que están definidas a nivel de modelo. Estas validaciones pueden ser longitud del campo excedida tipo de dato incorrecto etc.

```
 def create
   @fichero = Fichero.new(fichero_params)

   respond_to do |format|
     if @fichero.save
       format.html { redirect_to @fichero, notice: 'Fichero was successfully
created.' }
       format.json { render :show, status: :created, location: @fichero }
     else
       format.html { render :new }
       format.json { render json: @fichero.errors, status:
:unprocessable_entity }
     end
   end
 end
```

Método update

Este método modifica los atributos del objeto @fichero en función de los parámetros recibidos.

Nuevamente al igual que el método create se valida si la modificación tuvo éxito.
Si tuvo éxito se mostrará un mensaje flash avisando de que el fichero fue modificado correctamente. Si por el contrario hubo algún problema al realizar el grabado se mostrará un mensaje de error.

Los fallos que pueden provocar el rechazo de la modificación de objetos son los mismos que los comentados en el método create.

```
 def update
   respond_to do |format|
     if @fichero.update(fichero_params)
       format.html { redirect_to @fichero, notice: 'Fichero was successfully
updated.' }
       format.json { render :show, status: :ok, location: @fichero }
     else
       format.html { render :edit }
```

```
       format.json { render json: @fichero.errors, status:
:unprocessable_entity }
     end
   end
 end
```

Método destroy

Este método elimina el objeto @fichero es decir borra este registro del modelo Fichero. AL igual que los métodos explicados anteriormente, se valida si la eliminación tuvo existo o no devolviendo el correspondiente mensaje.

```
 def destroy
   @fichero.destroy
   respond_to do |format|
     format.html { redirect_to ficheros_url, notice: 'Fichero was
successfully destroyed.' }
     format.json { head :no_content }
   end
 end
```

Estos métodos que se han expuesto, como se ha comentado anteriormente son comunes para los tres controladores iniciales del sistema: EstandarsController, FicherosController y ResultadosController.

A continuación, se pasa a comentar las vistas que Rails genera por defecto cuando ejecutamos un scaffold.

VISTAS

_form.html.erb
En esta vista encontramos detalles importantes como por ejemplo la dirección a donde se enviará la información del formulario, en este caso colocamos fichero_create_path. Por otro lado, vemos como el formulario tiene los campos del fichero.

new.html.erb
Esta vista renderiza la vista anterior del formulario. La función submit del botón de envío del formulario es la encargada de valorar si es un alta nueva o un update

index.html.erb
Esta vista muestra los registros almacenados en la tabla. Es en ella donde se realiza el render de los datos.

En el siguiente punto se pasa a comentar otro elemento característico de los proyectos de Rails, los helpers.

HELPERS

En el desarrollo de proyectos con Ruby on Rails se permite la inyección de código Ruby en las vistas.
Esto es algo muy útil pero también ensucia el código y dificulta a veces su compresión ya que se mezcla código HTML con código Ruby para reducir esta inyección de código se diseñaron los helpers.

Un Helper es un módulo que ayuda a que las vistas no contengan demasiado código Ruby, ya que en un sistema MVC la idea es que la vista (V) sea tan simple como sea posible.

Por ello un Helper almacenará métodos que serán llamados desde las vistas, eliminando así gran parte del código Ruby que existiría en caso contrario.

Para el desarrollo de esta aplicación fue necesario el uso de un Helper en la vista de resultados, pero en los siguientes apartados se comentará más en profundidad la función de este helper.

A partir de este punto se pasa a explicar de forma detallada la parte más compleja del sistema que es donde va la funcionalidad principal de la validación de la diapositiva.

6.2.3 Núcleo de funcionalidad

En los siguientes apartados se explicarán cada una de las partes o módulos donde reside la complejidad del sistema.

Volviendo al objetivo del proyecto, este sistema tiene que permitir la inserción o subida de un fichero al sistema. Una vez este fichero se encuentra en el sistema se pasará a la fase de validación del mismo mediante la comprobación de los estándares almacenados previamente en base de datos. Según se vayan comprobando los estándares se irá almacenando en base de datos cada uno de los resultados. Finalmente se mostrará al usuario un resumen de los resultados obtenidos junto a la puntuación.

Este sería un resumen a grandes rasgos de la funcionalidad del sistema en los siguientes puntos se explicará cada una de las fases del sistema de forma clara y detallada.

Creación nuevo controlador

Para controlar la lógica referente a la validación del fichero se ha generado un controlador adicional llamado ArchivosController,

En este controlador reside toda la lógica principal de la inserción y validación de la diapositiva.

Para generar este nuevo controlador se ejecutó el siguiente comando:

```
rails g controller archivos subir_archivos
```

Como se puede observar en el comando anterior hemos definido el método subir_archivos directamente en la creación del controlador.

Antes de profundizar en la lógica que sigue este método es necesario hacer un inciso sobre el análisis de la entrada, en este caso el análisis de un fichero XML. Este inciso es muy necesario para comprender la lógica que se sigue en el método antes nombrado.

Analisis entrada

La entrada será un fichero en formato XML. ¿Por qué XML se ha elegido y no HTML? El código XML fue diseñado para transportar y almacenar datos y por el contrario el HTML fue diseñado para mostrar los datos. XML separa los datos de HTML y simplifica el intercambio y transporte de datos ya que almacena los datos en formato de texto plano además proporciona una manera en software y hardware-independiente de almacenamiento de datos. En próximas mejoras se podrá valorar si es necesario adaptar el sistema para la recepción de un fichero HTML.

Para analizar el fichero de entrada XML es necesario parasearlo para poder acceder a la información que es necesaria para la validación de cada uno de los estándares.

El fichero se analizará y se parseara por medio de la técnica ***scraping*** *‘rascar’*. El scraping es una técnica muy potente cuyo objetivo es la extracción de información por medio de la detección de etiquetas. Se utilizará esta técnica para detectar aquellas etiquetas que son necesarias para para validar los estándares de lectura fácil.

La característica más importante a destacar sobre un fichero XML desde el punto de vista de la técnica de ***scraping*** es su estructura de árbol. Al tener esta estructura de árbol es muy fácil distinguir los diferentes niveles de los que está constituido el fichero, esto hace que sea más fácil el análisis de las estructuras porque se reconocen a simple vista cada uno de los niveles. A partir de la ‘rama’ principal se puede ir navegando por cada una de las ramas secundarias. Ejemplo:

```
<Company>
 <Employee>
    <FirstName>Tanmay</FirstName>
    <LastName>Patil</LastName>
     <Address>
        <City>Bangalore</City>
        <State>Karnataka</State>
        <Zip>560212</Zip>
    </Address>
 </Employee>
</Company>
```

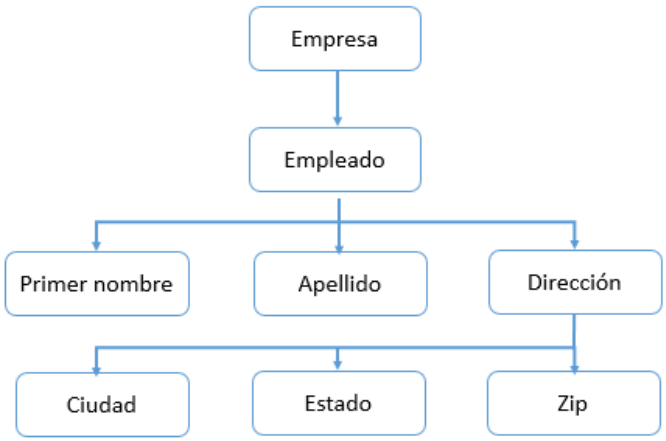


Ilustración 25 - Árbol XML 1

A continuación, se muestra el ejemplo de una diapositiva en formato XML

ESTRUCTURA FICHERO EJEMPLO XML (slide)

```
<pkg:package
    <pkg:part pkg:name="/_rels/.rels"…/>
    <pkg:part pkg:name="/ppt/slides/_rels/slide1.xml.rels"…/>
    <pkg:part pkg:name="/ppt/_rels/presentation.xml.rels" …/>
    <pkg:part pkg:name="/ppt/presentation.xml"…/>
  <pkg:part pkg:name="/ppt/slides/slide1.xml"
        <pkg:xmlData>
            <p:sld
          <p:cSld>
                <p:spTree>
                 <p:nvGrpSpPr…/>
                 <p:grpSpPr…/>
      <p:sp>
       <p:nvSpPr…/>
       <p:spPr…/>
       <p:txBody>
                          <a:bodyPr …>
                          <a:p>
                            <a:r>
          <a:rPr lang="es-ES_tradnl" …>
            <a:latin typeface="Arial" …/>
            <a:cs typeface="Arial"…/>
          </a:rPr>
                                  <a:t>SUBRAYADO</a:t>
                                 </a:r>
                           <a:endParaRPr lang="es-ES"…>
          <a:latin typeface="Arial" …/>
```

```
        <a:cs typeface="Arial" …/>
      </a:endParaRPr>
    </a:p>
  </p:txBody>
</p:sp>
```

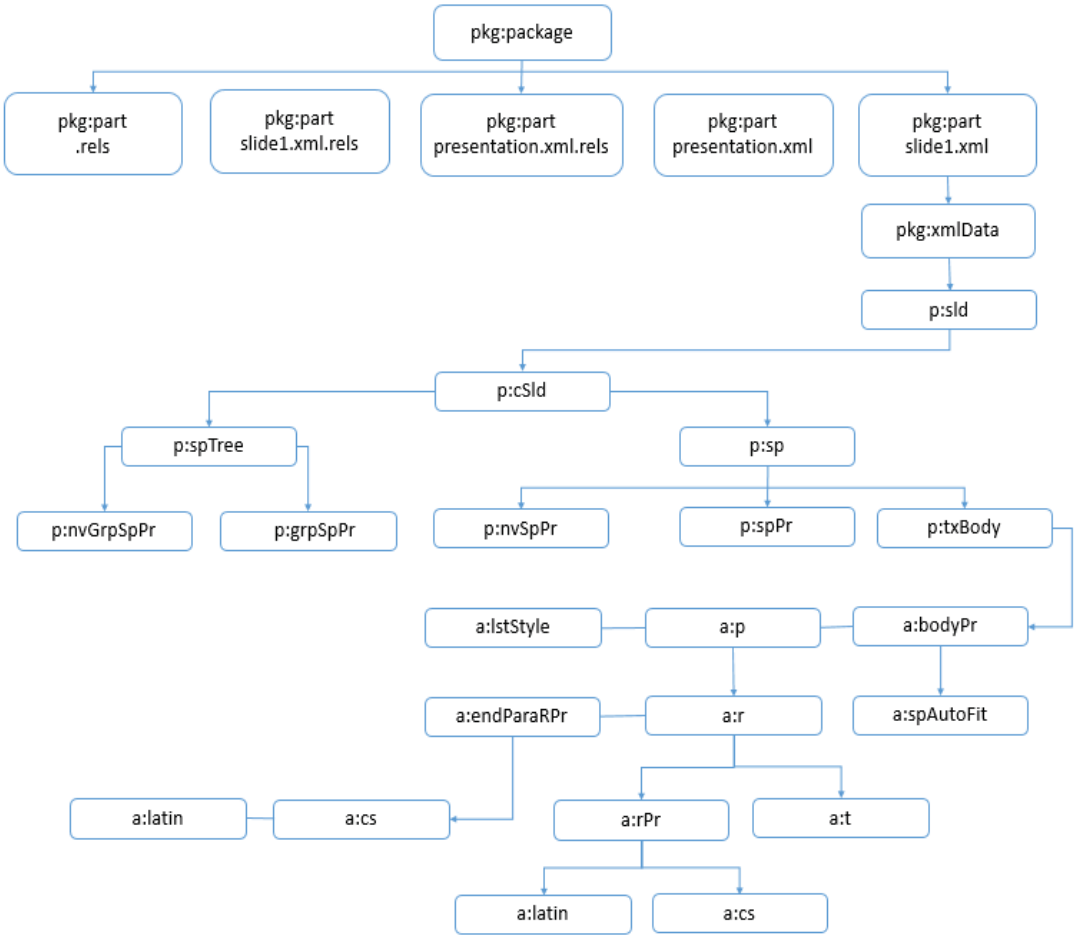


Ilustración 26 - Árbol XML 2

RESUMEN DE ETIQUETAS UTILIZADAS EN EL SCRAPING

ID	TIPOGRAFIA	ESTANDAR	DESCRIPCION	NIVEL 1	NIVEL 2	NIVEL 3	NIVEL 4	NIVEL 5	NIVEL 6	NIVEL 7	NIVEL 8	NIVEL 9	NIVEL 10	NIVEL 11	NIVEL 12	NIVEL 13	NIVEL 14
#1	TIPOGRAFIA	Fuentes utilizadas	Validar que el tipo de fuente del texto pertenece a los estilos aceptados	pkg:part	pkg:name="/ppt/slides/slide1.xml"	<pkg:xmlData>	<p:sld	<p:cSld>	<p:spTree>	<p:sp>	<p:txBody>	<a:p>	<a:r>	<a:latin	typeface		
#2	TIPOGRAFIA	Tamaño fuente	El tamaño del texto tiene que ser como mínimo 12 y como máximo 16	pkg:part	pkg:name="/ppt/slides/slide1.xml"	<pkg:xmlData>	<p:sld	<p:cSld>	<p:spTree>	<p:sp>	<p:txBody>	<a:p>	<a:r>	<a:rPr	sz		
#3	TIPOGRAFIA	Cursiva	Validar que no existe texto en cursiva	pkg:part	pkg:name="/ppt/slides/slide1.xml"	<pkg:xmlData>	<p:sld	<p:cSld>	<p:spTree>	<p:sp>	<p:txBody>	<a:p>	<a:r>	<a:rPr	i		
#4	TIPOGRAFIA	Negrita	Validar que no existen 'muchos' textos en negrita	pkg:part	pkg:name="/ppt/slides/slide1.xml"	<pkg:xmlData>	<p:sld	<p:cSld>	<p:spTree>	<p:sp>	<p:txBody>	<a:p>	<a:r>	<a:rPr	b		
#5	TIPOGRAFIA	Subrayado	Validar que no existen 'muchos' textos en subrayado	pkg:part	pkg:name="/ppt/slides/slide1.xml"	<pkg:xmlData>	<p:sld	<p:cSld>	<p:spTree>	<p:sp>	<p:txBody>	<a:p>	<a:r>	<a:rPr	u		
#6	TIPOGRAFIA	Sombreado	Validar que no existen textos con sombreado	pkg:part	pkg:name="/ppt/slides/slide1.xml"	<pkg:xmlData>	<p:sld	<p:cSld>	<p:spTree>	<p:sp>	<p:txBody>						
#7	TIPOGRAFIA	Mayuscula	Validar que no existen más de un texto en mayúsculas	pkg:part	pkg:name="/ppt/slides/slide1.xml"	<pkg:xmlData>	<p:sld	<p:cSld>	<p:spTree>	<p:sp>	<p:txBody>	<a:p>	<a:r>	<a:t>	RUBY		
#8	TIPOGRAFIA	Color fuente	Validar color NEGRO	<pkg:part	pkg:name="/ppt/slides/slide1.xml"	<pkg:xmlData>	<p:sld	<p:cSld>	<p:spTree>	<p:sp>	<p:txBody>	<a:p>	<a:r>	<a:rPr	<a:solidFill>	<a:srgbClr	val=
#9	Diseño	Color fondo diapositiva	Validar color BLANCO sólido	<pkg:part	pkg:name="/ppt/slides/slide1.xml"	<pkg:xmlData>	<p:sld	<p:cSld>	<p:bg>	<p:bgPr>	<a:solidFill>	<a:schemeClr	val=				
#11	Diseño	Cantidad palabras	Validar que la cantidad de palabras en la diapositiva no supere el límite establecido	<pkg:part	pkg:name="/docProps/app.xml"	<pkg:xmlData>	<Properties	<Words>									

Ilustración 27 - Etiquetas Scraping

En la tabla anterior se muestra el esquema de etiquetas que son necesarias para recuperar aquella información que permita validar los estándares. Es decir, en la tabla se muestra el recorrido de etiquetas que hay que seguir para llegar a la información que se busca. La etiqueta 'final' está marcada en verde. Por etiqueta final se entiende que es aquella etiqueta donde reside la información requerida para la validación del estándar.

Una vez entendida la tabla anterior y sabiendo que etiquetas tenemos que extraer, surge la pregunta de cómo es posible extraer esta información desde el proyecto de Ruby on Rails en el que se está trabajando.

Para poder parsear un fichero Ruby on Rails cuenta con una gema disponible llamada Nokogiri. A continuación, se explica detalladamente esta gema, su funcionalidad, su uso etc.

Gema Nokogiri

Nokogiri es una gema que permite el scrapero de ficheros es decir nos permite hacer búsquedas usando selectores CSS y XPath.

Para poder utilizar Nokogiri en el proyecto es necesario instalar la gema previamente. Para su instalación basta con añadir al Gemfile la siguiente línea y ejecutar bundle install.

```
gem install nokogiri
```

¿Como decimos a Nokogiri que fichero parsear?
La gema Nokogiri cuenta con varios métodos o formas de capturar el código que se quiere scrappear. A continuación, se describe cada una de ellas.

Desde String

`html_doc = Nokogiri::HTML("<html><body><h1>Mr. Belvedere Fan Club</h1></body></html>")`
 	
`xml_doc = Nokogiri::XML("<root><aliens><alien><name>Alf</name></alien></aliens></root>")`

Tabla 31 - Nokogiri 1

Desde Fichero
`doc = File.open("blossom.xml") { |f| Nokogiri::XML(f) }`

Tabla 32 - Nokogiri 2

Desde Internet
`require 'open-uri'` `doc = Nokogiri::HTML(open("http://www.threescompany.com/"))`

Tabla 33 - Nokogiri 3

Búsquedas básicas

Código de ejemplo

```
[shows.xml]
<root>
 <sitcoms>
   <sitcom>
     <name>Married with Children</name>
     <characters>
       <character>Al Bundy</character>
       <character>Bud Bundy</character>
       <character>Marcy Darcy</character>
     </characters>
   </sitcom>
   <sitcom>
     <name>Perfect Strangers</name>
     <characters>
       <character>Larry Appleton</character>
       <character>Balki Bartokomous</character>
     </characters>
   </sitcom>
 </sitcoms>
 <dramas>
   <drama>
     <name>The A-Team</name>
     <characters>
       <character>John "Hannibal" Smith</character>
       <character>Templeton "Face" Peck</character>
       <character>"B.A." Baracus</character>
       <character>"Howling Mad" Murdock</character>
     </characters>
   </drama>
 </dramas>
</root>
```

```
@doc = Nokogiri::XML(File.open("shows.xml"))
@doc.xpath("//character")
# => ["<character>Al Bundy</character>",
#     "<character>Bud Bundy</character>",
#     "<character>Marcy Darcy</character>",
#     "<character>Larry Appleton</character>",
#     "<character>Balki Bartokomous</character>",
#     "<character>John \"Hannibal\" Smith</character>",
#     "<character>Templeton \"Face\" Peck</character>",
#     "<character>\"B.A.\" Baracus</character>",
#     "<character>\"Howling Mad\" Murdock</character>"]
```

En el ejemplo anterior se puede ver cómo se recupera el contenido encerrado dentro de la etiqueta <character> </character> por medio de xpath

```
@doc.css("dramas name").first # => "<name>The A-Team</name>"
@doc.at_css("dramas name")    # => "<name>The A-Team</name>"
```

En este ejemplo se observa cómo se puede recuperar la información a través de etiquetas css.

Aquí se pone fin a la explicación sobre la gema Nokogiri, a continuación, se explicará el método *subir_archivos* del controlador ArchivosController.

Método subir_archivos

En este apartado se explicará en profundidad la funcionalidad del método subir_archivos. Este método almacena tanto la lógica de inserción del fichero en el sistema como la lógica de validación del mismo.

La estructura que sigue este método de forma genérica sería:

1. Recepción de fichero introducido por el usuario
2. Validación del fichero
3. Inserción del fichero en el sistema
4. Parseo del fichero a través de la gema Nokogiri
5. Recuperación de estándares desde base de datos
6. Validación de cada estándar
7. Generación de resultado

En este momento se empieza a explicar el código de forma detallada:

El primero paso es recoger el fichero introducido por el usuario en el campo de tipo input de la vista principal.

Este fichero se recupera por medio de los params. una vez que se obtiene el objeto 'archivo' se extrae su nombre y se graba en la variable global @nombrefile.

```
archivo = params[:archivo];
 #Nombre original del archivo.
 nombre = archivo.original_filename;
 @nombrefile = archivo.original_filename;
```

Una vez que se tiene localizado el archivo introducido por el usuario, se procede a almacenarlo dentro del sistema. En esta ocasión sea grabado en una variable de tipo 'constante' la ruta donde se van a ir almacenando los ficheros de los usuarios. Es importante destacar la necesidad de crear un directorio por cada uno de los ficheros subidos al sistema. Esto es importante para evitar fallos de concurrencia, es decir que varios usuarios inserten ficheros que son distintos en esencia, pero cuyo nombre coincide. Si se genera una carpeta por cada fichero cuyo nombre sea el dato exacto de la fecha con horas minutos segundo y hasta los nanosegundos se evitan estos fallos de concurrencia,

```
#Directorio donde se va a guardar.
 directorio = Ruta_directorio_archivos;
```

El siguiente paso es la validación del fichero. Este sistema tiene como requisito que únicamente acepta archivos en formato .XML. Por ello a continuación se muestra el código de validación del tipo de fichero.

```
    #Extensión del archivo.
    extension =
nombre.slice(nombre.rindex(".",nombre.length).downcase;
```

La variable extensión contiene la última parte de la cadena de la ruta del fichero insertado. Por ello una vez que se extrae el formato que tiene el fichero únicamente hay que validar si se corresponde con 'xml'. Normalmente las extensiones de los ficheros están escritas en minúsculas, pero por si acaso en algún momento estuviera en Mayúsculas se procede a su conversión con la función *donwcase*.

Cuando se valida que el formato es el adecuado el fichero se graba en el directorio creado.

El siguiente paso es crear el registro de este fichero en base de datos:

```
# Creacion del Fichero
         @fichero = Fichero.new
         @fichero.nombre = @nombrefile
         @fichero.created_at = Time.zone.today
```

```
        @fichero.save
        session[:fichero_id] = @fichero.id
```

En la creación del Fichero lo más importante es destacar la introducción de este fichero en una variable sesión. Esto se hace así porque será necesario poder acceder al fichero current desde la vista.

Las sesiones dentro de Rails se encuentran dentro de la gema actionpack. Esta gema se instala automáticamente cuando instalamos Rails. Cabe destacar que las sesiones en Rails se utilizan en el controlador y vistas.

```
   ## Metodo valida fichero
              # Declaracion de ARRAYS
         @msg_array = Array.new()
         @tips_fuentes = Array['Calibri','Arial', 'Candara',
'Corbel', 'Gill Sans', 'Helvética', 'Myriad', 'Segoe', 'Tahoma',
'Tiresias', 'Verdana']
         @tam_fuentes = Array.new()
         @cursiva_fuentes = Array.new()
         @bold_fuentes = Array.new()
         @u_fuentes = Array.new()
         @fuente_color = Array.new()
         @txt = Array.new()
```

Se generan los arrays que guardan los datos recogidos por Nokogiri:

`@msg_array`	Array que almacena los mensajes de error
`@tips_fuentes`	Almacena los diferentes tipos de letra que son permitidos por el sistema.
`@tam_fuentes`	Almacena los diferentes tamaños de letra que existen en la diapositiva
`@cursiva_fuentes`	Almacena aquellas palabras que estan en curvisa
`@bold_fuentes`	Almacena aquellas palabras que estan en negrita
`@u_fuentes`	Almacena aquellas palabras que estan en curvisa
`@fuente_color`	Almacena los diferentes tipos de letra que existen en la diapositiva
`@txt`	Almacena el total de palabras

Ahora se abre el fichero con Nokogiri para proceder a su parseo y extracción de datos.

```
 @doc = File.open("public/archivos/"+@fichero.nombre) { |f|
Nokogiri::XML(f) }

#Eliminamos los name space
    @doc.remove_namespaces!

# Titulo diapositiva
  @titulo = @doc.css('title')

# Total palabras
  @words = @doc.css('Words')
  @total_words= @words.map(&:text)[0]
```

En el siguiente bloque se extrae la información necesaria del fichero para poder validarlo contra los estándares. Esta información se recoge mediante las etiquetas que aparecían en la tabla que se mostró en capítulos anteriores.

```
 # TIPOGRAFIA

           @pptx =
@doc.css("part[name='/ppt/slides/slide1.xml']").css('xmlData').css('sl
d').css('cSld').css('spTree').css('sp').css('txBody').css('p').css('r'
).css("latin")
           @color_fondo =
@doc.xpath("//part[@name='/ppt/slides/slide1.xml']//xmlData//sld//cSld
//bg")

           @ppt_tipo =
@doc.xpath("//part[@name='/ppt/slides/slide1.xml']//xmlData//sld//cSld
//spTree//sp").each do |node|
             @fuente=node.xpath("//txBody//p//r//rPr//latin") #
sacamos la lista de fuentes
             @fuente_color =
node.xpath("//txBody//p//r//rPr//solidFill//srgbClr")
            end

           @ppt_tam =
@doc.xpath("//part[@name='/ppt/slides/slide1.xml']//xmlData//sld//cSld
//spTree//sp//txBody//p//r//rPr").each do |rpr|
             @tam_fuentes.push(rpr.attr('sz')) # sacamos la lista de
tamanio fuentes
             @cursiva_fuentes.push(rpr.attr('i')) # sacamos la lista
de cursiva_fuentes fuentes
```

```
              @bold_fuentes.push(rpr.attr('b')) # sacamos la lista de
negrita fuentes
              @u_fuentes.push(rpr.attr('u')) # sacamos la lista de
subraydo fuentes
             end

           @ppt_txt =
@doc.xpath("//part[@name='/ppt/slides/slide1.xml']//xmlData//sld//cSld
//spTree//sp//txBody//p//r//t").each do |t|
              @txt.push(t.text) # sacamos la lista de tamanio fuentes
           end
```

Una vez que se tiene toda la información necesaria para la validación en variables, se procede a la comprobación de cada uno de los estándares.

En este documento se van a explicar aquellos estándares que son diferentes en lo que se refiere al código de validación para no crear explicaciones repetitivas. El símbolo # en Rails sirve para comentar líneas.

Validación de fuentes

```
 # def estadar_tip1_fuente   .attr('typeface')

     #Recuperamos de base de datos el estandar con id = 1
          @estadr = Estandar.find_by(id:1)

           ban_fuente = false

     # Comprobamos que el array de fuentes no es vacío
           if @fuente != nil && @fuente.size > 0

# Recorremos cada una de las fuentes
           @fuente.each do |font|
          # Generamos un nuevo resultado
             @resultado = Resultado.new
# Pasamos el id del fichero que se está validando
             @resultado.fichero_id = @fichero.id
# Pasamos el id del estándar que se está validando
             @resultado.estandar_id = @estadr.id
     #Accedemos el atributo typeface que el que almacena la fuente en
el fichero XML.

             tipografia = font.attr('typeface')
#Buscamos la fuente en el array de fuentes válidas declarado
anteriormente @tips_fuentes
```

```
              font_find = @tips_fuentes.detect{|w| w == tipografía}
#Si el tipo de letra no existe en este array el estándar no se cumple
por lo que asignamos un '1' al resultado junto con el mensaje de error
correspondiente. Si el tipo de fuente está dentro de este array se
asigna el coderror '0' lo que significa que el estándar se cumple.

              if font_find.nil?
                ban_fuente = true
                @resultado.coderror = 1
                @resultado.msg_error = 'Fuente detectada no válida'+ '
' + tipografia
              else
                @resultado.coderror = 0
              end
# Grabamos el resultado
              @resultado.save
            end
            end
```

Validación de fuentes en negrita

```
  # def tip4_negrita .attr('b')
     #Recuperamos de base de datos el estándar con id = 4
     # Este estándar válida que no puede existir más de 5 palabras en
negrita dentro de la diapositiva.

            @estadr = Estandar.find_by(id:4)

          #Inicializamos el contador de negritas.
            cuenta_bold = 0
            ban_bold = false

     # Comprobamos que el array de negritas no es vacío
            if @bold_fuentes !=nil &&  @bold_fuentes.size > 0
            @bold_fuentes.each do |b|
              if !b.nil?
          #Incrementamos el contador
                cuenta_bold = cuenta_bold + 1
              end
#Generamos el nuevo resultado
              @resultado = Resultado.new
              @resultado.fichero_id = @fichero.id
              @resultado.estandar_id = @estadr.id
#Si se ha superado el límite de palabras en negrita se inserta
coderror '1' en el resultado creado junto con el mensaje de error
correspondiente. Si por el contrario no es superado se asigna el valor
'0' a coderror.
              if cuenta_bold > 5
                ban_bold = true
                @resultado.coderror = 1
```

```
                @resultado.msg_error = 'Superado el límite de cuadros
de texto en formato negrita'
              else
                @resultado.coderror = 0
              end
#Grabamos el resultado
              @resultado.save
            end
```

Validación color de fuente

```
#Color NEGRO tipografía

     #Recuperamos de base de datos el estándar con id = 5
#Este estándar estipula que el color de la fuente siempre ha de ser
negro.
           @estadr = Estandar.find_by(id:8)

     # Comprobamos que el array de @fuente_color no es vacío
           if @fuente_color !=nil && @fuente_color.size > 0

           ban_fuente_color = false

#Recorremos cada uno de los elementos de array
            @fuente_color.each do |font|
#Generamos un nuevo resultado con el id del fichero que se está
validando y con el id del estándar en cuestión.
              @resultado = Resultado.new
              @resultado.fichero_id = @fichero.id
              @resultado.estandar_id = @estadr.id

#Extraemos el color de elemento del array por medio del atributo 'val'
              color = font.attr('val')
#Validamos que el color sea diferente de 'negro'.
#Si es distinto se asigna coderror '1' y el mensaje de error
correspondiente. Si por el contrario es negro se asigna el coderror
'0'.
               if color != '000000'
                ban_fuente_color = true
                @resultado.coderror = 1
                @resultado.msg_error = 'Color de fuente detectado no
válido'+ ' ' + color
              else
                @resultado.coderror = 0
              end
#Grabamos el resultado
              @resultado.save
             end
```

Una vez se validan todos los estándares se hace la llamada a la página de visualización de los resultados, *resultados_path*:

```
respond_to do |format|
                format.html { redirect_to resultados_path,
notice: 'El fichero se proceso correctamente'}
                 format.json { render :show, status: :ok,
location: @fichero }
end
```

Una vez explicado el método de *subir_archivos*, se pasa a explicar el helper desarrollado para la asignación de puntuación al fichero. Este helper es el asociado a la vista de Resultados.

Esto se calcula de forma dinámica recorriendo la tabla Resultados. Una vez se obtiene la puntuación el helper envía el resultado a la vista y esta lo muestra.

El módulo *ResultadosHelpe*r implementa un método llamado valoración. Este método a grandes rasgos lo que hace es recuperar todos los resultados asociados a un Fichero. Una vez tiene todos los resultados los separa entre los resultados con error y los resultados sin error. Se realiza un conteo de ambos tipos de resultados y se inserta en un array, llamado puntuación []

```
module ResultadosHelper

 def valoracion

# Declaración del array
    puntuacion = []

#Recuperación de Resultados fallidos
   resultados_bad = Resultado.select(:estandar_id).where(:fichero_id
=> session[:fichero_id], :coderror => 1).group(:estandar_id)

#Recuperación de resultados correctos
   resultados_ok  = Resultado.select(:estandar_id).where(:fichero_id
=> session[:fichero_id], :coderror => 0).group(:estandar_id)

estandar_bad   = resultados_bad.map {|bad| bad.estandar_id }

resultados_ok = Resultado.select(:estandar_id).where('fichero_id = ?
AND  estandar_id NOT IN (?) AND coderror =
0',session[:fichero_id],estandar_bad).group(:estandar_id)
   estandar_ok    = resultados_ok.map {|ok| ok.estandar_id }

  # Inserción de resultados en base de datos
   puntuacion.push(estandar_ok.size)
```

```
   puntuacion.push(estandar_bad.size)

# Devolvemos el array
     return puntuacion
 end
end
```

Una vez visto el helper se procede a destacar los puntos más importantes de la vista de resultados. Los puntos más importantes son aquellas secciones de la vista que tiene código Ruby inyectado .

Aquí se muestra el fragmento de código donde se realiza la llamada al método 'valoración' del helper anteriormente descrito.

```
Recuperamos el valor devuelto por el método 'valoración' en el
variable global @puntuación. Accedemos al total de estándares
correctos, accediendo a la posición [0] del array @puntuacion. Para
acceder a los estándares incorrectos se accede a la posición [1] del
array @puntuacion.

<div class="jumbotron">
     <h1>Tu puntuación es:</h1>
      <%@puntuacion = valoracion%>
     <div id='detail_puntu'><%= @puntuacion[0].size%></div>
   </div>
    <p><h5>
      <span class="glyphicon glyphicon-ok"></span><b> Correctos:</b>
<%= @puntuacion[0]%>
      <span class="glyphicon glyphicon-remove"></span><b>
Incorrectos:</b><%= @puntuacion[1]%>
    </h5></p>
```

Para la visualización de los estándares con diferentes colores en base a si existen en errores o no para ese estándar se realiza desde código Ruby.

Ilustración 28 - Panel estándares

Se recuperan todos los estándares y se almacenan en la variable mis_estandares. Se recorren todos los objetos 'estándar' de la variable. Para cada estándar se comprueba si para el fichero que tenemos en session existe un resultado con error para ese estandar_id. Si no existen resultados erróneos el panel se pinta de color verde junto con el icono de validación. Si existe algún resultado erróneo el panel se pinta de color rojo junto con el icono del aspa.

```
<% mis_estandares = Estandar.all%>
 <div class="panel-group" id="accordion">
     <% mis_estandares.each do |mi_estandar| %>
      <% resultado_bad = Resultado.where(:fichero_id =>
session[:fichero_id], :estandar_id => mi_estandar.id,
:coderror=>1).count%>
   <% if resultado_bad == 0%>
   <div class="panel panel-success">
         <%else%>
       <div class="panel panel-danger">
    <%end%>
```

.
.
.

Exportación a Excel

Para realizar la exportación a Excel de los resultados se han seguidos los siguientes pasos:

config/application.rb
`require 'csv'`

index.hml.erb - Resultados
Añadimos el botón de descarga indicándole el formato en el que queremos que exporte <%= link_to resultados_path(format: "csv"), class: "btn btn-default" do %> Download <% end %>

resultados_controller.rb

```
def index
 @resultados = Resultados.order(:estandar_id)
 respond_to do |format|
   format.html
   format.csv { send_data @resultados.to_csv }
   format.xls # { send_data @resultados.to_csv(col_sep: "\t") }
 end
end
```

models/resultado.rb

```
def self.to_csv(options = {})
 CSV.generate(options) do |csv|
   csv << column_names
   all.each do |product|
     csv << product.attributes.values_at(*column_names)
   end
 end
end
```

config/initializers/mime_types.rb

```
Mime::Type.register "application/xls", :xls
```

views/resultados/index.xls.erb

- Vista de resultados para excel

```
<table border="1">
 <tr>
   <th>ID Resultado</th>
   <th>Estandar tipo</th>
   <th>Estandar descripcion</th>
   <th>Código Error</th>
   <th>Mensaje error</th>
 </tr>
 <% @resultados = Resultado.where(:fichero_id =>
session[:fichero_id])%>
 <% @resultados.each do |result| %>
 <tr>
   <td><%= result.id %></td>
   <td><%= Estandar.find(result.estandar_id).tipo %></td>
   <td><%= Estandar.find(result.estandar_id).descripcion %></td>
   <td><%= result.coderror %></td>
   <td><%= result.msg_error %></td>
 </tr>
 <% end %>
</table>
```

6.3 Pruebas

En cuanto a la depuración de código se utilizaron diferentes métodos. Por un lado, para realizar pruebas de depuración sencilla se utilizó el comando:

puts home

gets

Este comando te muestra en consola el contenido de la variable en este caso home y congela la ejecución hasta que se pulsa la tecla Intro.

Para la realización de depuración de código más profunda y compleja se utilizó la gema Byebug. Esta gema es la sucesora de la gema debugger. Esta gema es un depurador de código para Ruby y a través de la consola podemos para la ejecución en los diferentes puntos de ruptura que se pongan en el código.

Esta gema permite la depuración:

- Paso a paso
- Salto al siguiente punto de ruptura
- Evaluación de variables desde la consola

El modo de utilización de esta gema es colocando la palabra ‘byebug’ en las partes del código donde se quieran colocar los diferentes puntos de ruptura.

Para la realización de pruebas en el sistema se ha utilizado una gema de Rails que facilita esta tarea de forma considerable.

La gema en cuestión es Rspect.

RSpec es un marco de pruebas unitarias para el lenguaje de programación Ruby. RSpec es diferente de los marcos tradicionales como JUnit xUnit porque RSpec es una herramienta de desarrollo centrada en el comportamiento. Esto querie decir que RSpec en lo que se centra es en que la aplicación devuelva lo que se espera que tiene que devolver en base a los requisitos estipulados. RSpec está diseñado para que quede claro si el código de destino se está comportando correctamente, es decir después de la especificación.

Para utilizar esta gema hay que seguir el mismo procedimiento que se ha explicado anteriormente para la instalación de gemas, añadiendo la gema RSpec al Gemfile y ejecutando posteriormente el bundle install.

Todas las pruebas serán grabadas en un nuevo directorio generado llamado \spec\ .

Una vez generado el fichero de pruebas para ejecutarlo bastaría con lanzar el siguiente comando desde el terminal:

```
rspec spec spec\resultados_spec.rb
```

Cuando el comando se completa, debería ver una salida que se parece a esto:

```
Finished in 0.002 seconds (files took 0.11101 seconds to load)
1 example, 0 failures
```

Para validar la aplicación se generan pruebas de Rspec tanto a nivel de los modelos, a nivel de controlador y a nivel de helper. Los ficheros de pruebas se encuentran en el directorio como se ha indicado anteriormente.

Las siguientes líneas muestran un ejemplo de algunas de las pruebas realizadas mediante esta gema en el proyecto.

Edición Estándar

```
RSpec.describe Estandar, :type => :model do
  it "Update Estandar" do
    st_01 = Estandar.create!(tipo: "PRUEBA1", descripcion: "Mensaje de prueba")
    st_01.tipo = "PRUEBA2"
     expect(st_01.tipo).to eq("PRUEBA2")
  end
end
```

La prueba anterior crea un nuevo Estándar con unos atributos por defecto. Una vez el estándar está grabado en base de datos se procede a cambiar el valor de uno de estos atributos. La finalidad de la prueba es comprobar que el sistema permite la modificación de un estándar.

Creación nuevo Fichero

```
RSpec.describe Fichero, :type => :model do
  it "Nuevo Fichero" do
    lindeman = Fichero.create!(nombre: "PRUEBA1")
    chelimsky = Fichero.create!(nombre: "PRUEBA2")
     expect(lindeman.nombre).to eq("PRUEBA1")
```

```
    expect(chelimsky.nombre).to eq("PRUEBA2")
  end
end
```

La prueba anterior comprueba que el sistema permite y realiza de forma satisfactoria la creación de un nuevo Fichero.

Validacion Helper

```
RSpec.describe ResultadosHelper, :type => :helper do
 describe "Calculo de la valoracion" do
   it "Puntuación" do
      session[:fichero_id]
      chelimsky = Fichero.create!(nombre: "PRUEBA2")
      Resultado.create!(coderror: 0, msg_error: "Lindeman", fichero_id: chelimsky.id)
      Resultado.create!(coderror: 1, msg_error: "Lindeman", fichero_id: chelimsky.id)
      Resultado.create!(coderror: 1, msg_error: "Lindeman", fichero_id: chelimsky.id)
      session[:fichero_id] = chelimsky.id
     expect(helper.valoracion.size).to eq 2
   end
 end
```

Esta última prueba que se muestra valida el comportamiento del helper de la vista de Resultados. La prueba trata de generar una serie de resultados ficticios para un fichero previamente creado y una vez esta todo dado de alta ejecuta la función almacenada en el helper ‘valoración’. La prueba comprueba si el valor devuelto por validación tiene tamaño dos. Es decir, si efectivamente está devolviendo dos valores uno a asociado a los resultados correctos y el otro a los resultados incorrectos.

A continuación, se expone la pila de pruebas utilizada para cada modelo y controlador:

Validación de modelo y controlador Estándar

```
RSpec.describe Estandar, :type => :model do
  it "Nuevo Estandar" do
    lindeman = Estandar.create!(tipo: "PRUEBA1", descripcion:
"Lindeman")
    chelimsky = Estandar.create!(tipo: "PRUEBA2", descripcion:
"Chelimsky")
    expect(lindeman.tipo).to eq("PRUEBA1")
    expect(chelimsky.tipo).to eq("PRUEBA2")
  end
end

RSpec.describe Estandar, :type => :model do
  it "Update Estandar" do
    st_01 = Estandar.create!(tipo: "PRUEBA1", descripcion:
"Mensaje de prueba")
    st_01.tipo = "PRUEBA2"
     expect(st_01.tipo).to eq("PRUEBA2")
  end
end

RSpec.describe EstandarsController, :type => :controller do
  describe "GET #index" do
    it "responds successfully with an HTTP 200 status code" do
      get :index
      expect(response).to be_success
      expect(response).to have_http_status(200)
    end

    it "renders the index template" do
      get :index
      expect(response).to render_template("index")
```

```
    end

    it "loads all of the estandars into @posts" do
      st1, st2 = Estandar.create!, Estandar.create!
      get :index
      expect(assigns(:estandars)).to match_array([st1, st2])
    end
  end
```

Validación de modelo y controlador Fichero

```
RSpec.describe Fichero, :type => :model do
  it "Update Fichero" do
    st_01 = Fichero.create!(nombre: "PRUEBA1")
    st_01.nombre = "PRUEBA2"
     expect(st_01.nombre).to eq("PRUEBA2")
  end
end


RSpec.describe FicherosController, :type => :controller do
  describe "GET #index" do
    it "responds successfully with an HTTP 200 status code" do
      get :index
      expect(response).to be_success
      expect(response).to have_http_status(200)
    end

    it "renders the index template" do
      get :index
```

```
      expect(response).to render_template("index")
    end

    it "loads all of the Ficheros into @posts" do
      st1, st2 = Fichero.create!, Fichero.create!
      get :index
      expect(assigns(:ficheros)).to match_array([st1, st2])
    end
  end
```

Validación de modelo y controlador Resultado

```
RSpec.describe Resultado, :type => :model do
  it "Nuevo Resultado" do
    lindeman = Resultado.create!(coderror: 0, msg_error:
"Lindeman")
    chelimsky = Resultado.create!(coderror: 1, msg_error:
"Chelimsky")
    expect(lindeman.msg_error).to eq("Lindeman")
    expect(chelimsky.msg_error).to eq("Chelimsky")
  end
end

RSpec.describe Resultado, :type => :model do
  it "Update Resultado" do
    st_01 = Resultado.create!(coderror: 0, msg_error: "Mensaje
de prueba")
    st_01.coderror = 1
     expect(st_01.coderror).to eq(1)
  end
end
```

```
RSpec.describe ResultadosController, :type => :controller do
  describe "GET #index" do
    it "responds successfully with an HTTP 200 status code" do
      get :index
      expect(response).to be_success
      expect(response).to have_http_status(200)
    end

    it "renders the index template" do
      get :index
      expect(response).to render_template("index")
    end

    it "loads all of the Resultados into @posts" do
      st1, st2 = Resultado.create!, Resultado.create!
      get :index
      expect(assigns(:resultados)).to match_array([st1, st2])
    end
  end
```

6.4 Datos adicionales sobre el proyecto

El proyecto se encuentra publicado en Heroku en la siguiente en la dirección: https://prue555.herokuapp.com/

El código fuente del mismo esta subido en la plataforma Github en el siguiente repositorio: https://github.com/San22c/w.easy

7. Conclusiones

Hoy en día las nuevas tecnologías son un pilar fundamental en muchos campos, pero sobre todo se quiere destacar la importancia de estas en el ámbito educativo y en el ámbito de la comunicación.

Estas tecnologías ayudan a mejorar la educación de las personas proporcionándolas nuevas herramientas para el estudio. Se ha pasado de estudiar únicamente de los libros de texto a estudiar desde recursos tecnológicos como pueden ser aplicaciones web o móviles, programas de escritorio, uso de presentaciones PowerPoint, etc.

Aunque estos cambios a priori son para mejorar la educación es cierto que muchos de ellos no están pensados en un primer momento para todos los públicos, es decir al igual que pasaba con los libros de texto es necesario hacer adaptaciones para que personas con ciertas discapacidades puedan acceder a este conocimiento que se trata de impartir.

El acceso a la información y al conocimiento es un derecho que por ley tienen todas las personas. El problema es que la sociedad no es consciente de que hay ciertos colectivos de personas que posee algún tipo de discapacidad que les impide acceder a este tipo de información porque el material en el que se expone no está adaptado para ellos. Por lo que se está violando un derecho fundamental de las personas.

Es necesario coger conciencia de este problema que afecta en muchos ámbitos de las nuevas tecnologías. Normalmente cuando se desarrolla una aplicación no se tienen en cuenta las necesidades (en el ámbito de la discapacidad) que requieren ciertos colectivos de personas.

Unos ejemplos de ello serian páginas web que no están adaptar para personas con discapacidad visual. Estas personas necesitarían que la web este adaptada con un lector que vaya narrando el contenido de la misma, cosa que la gran mayoría no lo proporciona. Otro ejemplo seria personas con daltonismo, muchas webs o aplicaciones no están pensadas para personas con este tipo de discapacidad y utilizan códigos de color sin ningún otro tipo de identificación por lo que para estas personas es imposible acceder a la información.

Este trabajo se centra principalmente en mejorar un tipo de recurso educativo como son las diapositivas, para que estas sean accesibles para el colectivo con discapacidad intelectual. Esto surge porque las diapositivas a día de hoy son un recurso utilizado de forma habitual en la impartición de docencia. Por ello surge la necesidad de desarrollar un software que valide si una diapositiva es apta para personas con este tipo de discapacidad.

Este proyecto se ha desarrollado bajo la tecnología de Ruby on Rails empleando como entorno de desarrollo el editor de texto Atom. Esta elección fue analizada y valorara en capítulos anteriores y resulto muy adecuada para el desarrollo del sistema.

Se quiere destacar la importancia de las diferentes fases que se han ido completando a lo largo de la vida de este proyecto. La fase de primer análisis del problema que se planteaba, la fase de documentación sobre el tema a tratar en este caso la discapacidad intelectual. Por otro lado, se quiere destacar la importancia de una buena toma de requisitos ya que es fundamental para realizar un desarrollo del sistema que satisfaga todas las necesidades demandas por el cliente, en este caso el tutor del proyecto. También cabe destacar el diseño del sistema analizando previamente cada aspecto relevante, definiendo cada uno de los módulos que intervenían en el así como la arquitectura del sistema en su conjunto.

Una vez se han completado de forma satisfactoria cada una de las fases nombradas anteriormente, se tienen todas las bases necesarias para poder realizar la implementación del sistema.

Al abordar este proyecto se toma consciencia de lo importante que es hacer accesible la información para cualquier persona, por ello surge el interés por conocer más acerca sobre los diferentes proyectos que existen hoy en día para hacer más accesible la información, como es el proyecto o metodología de lectura fácil. Es un tema realmente interesante y que todavía no se trata bajo mi punto de vista con la importación que se merece.

Por ello a través de este proyecto se pretende realizar una pequeña aportación para hacer más accesible la información y el conocimiento a todo el mundo.

En cuanto al aspecto personal, la realización de este proyecto ha servido para profundizar en un gran conjunto de conocimientos que se han adquirido a lo largo de la carrera. Aplicando estos conocimientos a un caso ‘real’ en el que se partía de un proyecto de cero. También la realización de este proyecto ha servido para valorar la importancia de una buena gestión del proyecto, así como el ser conscientes de lo que es enfrentarte a abordar un proyecto y cumplir unos plazos.

Este proyecto por otro lado también ha servido para afianzar y demostrar los conocimientos que se han aprendido en un ámbito laboral, ya que se ha sido de gran ayuda contar con experiencia laboral previamente a la realización de este proyecto. Esta experiencia te aporta en este caso el abordar todo siempre desde varias perspectivas valorando cual sería la mejor opción y por qué. También sobre el modo de desarrollado de forma ordenada, clara, sencilla y comentando cada uno de los métodos implementados.

Aunque el proyecto no fue extremadamente difícil de abordar es cierto que en las diferentes fases del proceso surgieron ciertos problemas o conflictos. A destacar por ejemplo problemas con la compatibilidad de ciertas gemas con la versión de Rails que en un inicio se había instalado, declaración de nombres de atributos con codificación de camello cuando el gestor de base de datos elegido PostgreSQL no admite este tipo de codificación etc. Pero, aunque han surgido diferentes conflictos ninguno se ha llevado un tiempo de corrección excesivo.

Para concluir se quiere resaltar la importancia de hacer desarrollos siempre pensando en el colectivo al que estará dirigido e intentar hacer los desarrollos lo más versátiles posibles para poder asumir mejorar o modificaciones de cara a futuras necesidades.

También destacar lo importante que es pararse a pensar en si lo que estamos desarrollando será válido para todas las personas o si se les presta las ayudas necesarias para que su experiencia de usuario sea plenamente satisfactoria y no discriminatoria.

Sobre la realización de este proyecto en esencia ha sido una experiencia muy enriquecedora y que ha aportado una gran satisfacción al realizar con éxito la gestión y el desarrollo del producto final cumpliendo con todos los requisitos que se estipularon previamente.

8. Futuras Líneas

Una vez finalizado el desarrollo, la aplicación cumple con los requisitos establecidos al inicio del proyecto. Si bien es cierto que existen diversos aspectos a mejorar y mucha funcionalidad que se le podría agregar.

Un aspecto muy importante a mejorar de cara a futuras versión seria ofrecer la posibilidad de aceptar directamente un tipo de archivo con formato pptx es decir la posibilidad de aceptar una presentación sin necesidad de que el usuario la convierta previamente a XML.

Esta mejora va de la mano con la posibilidad de analizar no solo una diapositiva si no realizar la validación de la presentación en su conjunto.

También sería interesante agregar la funcionalidad de que el propio sistema te devuelva exactamente el texto que incumple un estándar y no solo mostrar el mensaje genérico de que falla porque no se cumple el estándar.

Al hilo de la validación una mejora a tener en cuenta seria la creación de una vista para usuarios 'Administradores' desde la cual se puedan dar de alta, modifica e eliminar o dar de baja los diferentes estándares. Así como realizar tareas de reportes de datos sobre la aplicación como pudiera ser control de usuarios que acceden al sistema. Esto se podría realizar implementando en la aplicación de la gema *impresionist*. Esta gema recoge toda la información sobre acciones que se realizan en la aplicación.

Finalmente, en cuanto al apartado de las exportaciones como mejora se podría dar la opción de exportación a PDF, así como el envío de los resultados vía email.

9. Bibliografía

Presentación PowerPoint

[1] Rubén Mesía Maraví, EL EMPLEO DIDÁCTICO DE LAS DIAPOSITIVAS EN POWER POINT ,Recuperado de http://docplayer.es/5303434-El-empleo-didactico-de-las.html

[2] Evelio Martínez, (2012), Por qué las diapositivas en una presentación pueden suprimir la información oral, Recuperado de https://eveliomartinez.wordpress.com/2012/02/21/por-que-las-diapositivas-en-una-presentacion-pueden-suprimir-la-informacion-oral/

[3] Marta Molina, María C. Cañadas e Isidoro Segovia, (2013), LAS DIAPOSITIVAS COMO APOYO AL DISCURSO ORAL EN LA DOCENCIA UNIVERSITARIA. PERSPECTIVA DE LOS ESTUDIANTES EN EL MARCO DE UN PROCESO DE MENTORIZACIÓN, Recuperado de http://www.ugr.es/~recfpro/rev173COL10.pdf

Discapacidad intelectual

[4] NICHCY, (2010), Discapacidades Intelectuales, Recuperado por http://www.parentcenterhub.org/wp-content/uploads/repo_items/spanish/fs8sp.pdf

[5] National Center on Birth Defects and Developmental Disabilities, Facts About Intellectual Disability, Recuperado por https://www.cdc.gov/ncbddd/actearly/pdf/parents_pdfs/IntellectualDisability.pdf

[6] American Association of Intellectual and Developmental Disabilities; National Center on Birth Defects and Developmental Disabilities; the United Nations Development Program; and the Centers for Disease Control and Prevention, ¿Qué es la discapacidad intelectural?, Recuperado por http://www.specialolympics.org/RegionsPages/content.aspx?id=29901&LangType=1034

[7] Mac Perez Lopez,(2011), La discapacidad intelectual, causas, diagnóstico y signos, Recuperado por http://www.discapacidadonline.com/discapacidad-intelectual-causas-diagnostico-signos.html

[8] http://www.plenainclusion.org/sites/default/files/lectura-facil-metodos.pdf

[9] University of Hertfordshire, (2017), Recuperado por http://www.intellectualdisability.info

[10] Cermi, Recuperado por http://www.cermi.es/es-ES/orientacion/Paginas/Inicio.aspx

[11] Unidad de Promoción y Desarrollo La Salina, Recuperado por http://www.dipsanet.es/upd/pdfs/16_DACApoyoMetodologicoDocentes/PautasDocentesNecesidadesEspeciales.pdf

Tecnologías

[12] Sierra Urrecho, A. (1999), *Programación en C / C++*, Madrid, España

[13] Ceballos, Fco, (2010), *JAVA 2 Curso de programación*, Madrid, España

[14] Carlson, L., Richardson, L. (2066*), Ruby Cookbook*

[15] TIOBE software BV, Recuperado por https://www.tiobe.com/tiobe-index/ruby/

[16] TXEMA RODRIGUEZ , (2015), Recuperado por https://www.xataka.com/aplicaciones/20-anos-de-java-celebramos-su-tremenda-influencia-en-el-mundo-del-software-y-la-programacion

[17] Hoy en Ciencia, Recuperado por https://www.genbetadev.com/actualidad/hoy-en-ciencia-nace-el-lenguaje-de-programacion-c

[18] Rubén Velasco, Recuperado por https://www.redeszone.net/2015/06/29/atom-el-editor-de-texto-de-github-alcanza-la-version-estable-1-0/

[19] Ruby El mejor amigo de un desarrollador, Recuperado por https://www.ruby-lang.org/es/

[20] Microsoft, Visual Studio, Recuperado de https://www.visualstudio.com

[21] Oracle, Java, Recuperado de https://www.java.com

[22] Rafa Elma, PostgreSQL, Recuperado de http://www.postgresql.org.es/

[23] Oracle, MySQL, Recuperado de https://www.mysql.com/

[24] Microsoft, SQL Server, Recuperado de https://www.microsoft.com/es-es/sql-server/sql-server-2016

[25] EcuRed,Recuperado en https://www.ecured.cu/Sistema_Gestor_de_Base_de_Datos

[26] Pedro Salinas en https://users.dcc.uchile.cl/~psalinas/uml/casosuso.html

[27] Oposicionestic, en https://oposicionestic.blogspot.com.es/2011/06/arquitectura-cliente-servidor.html

[28] Dr. Manuel Blázquez Ochando en http://ccdoc-basesdedatos.blogspot.com.es/2013/02/modelo-entidad-relacion-er.html

[29] Cohero en http://codehero.co/ruby-on-Rails-pruebas-unitarias/

[30] Jose Manuel Ayala Wilson en http://jmaw.blogspot.com.es/2013/01/arquitectura-de-aplicaciones-web-capa.html

[31] Julio Giampiere Grados Caballero en https://devcode.la

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Thu Jun 08 23:11:33 CEST 2017
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)