E.T.S. de Ingeniería Industrial, Informática y de Telecomunicación

Plataforma web de asistencia en el diseño e implementación de formularios HTML dinámicos basados en reglas

Grado en Ingeniería informática

Trabajo Fin de grado

Haizea Martin Alzueta

Iñigo Fermin Ezcurdia

Pamplona, 3 de junio de 2021

upna

Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Agradecimientos

En primer lugar, quiero agradecer a mi familia ya que con su esfuerzo y dedicación me ayudaron a culminar mi carrera universitaria y me dieron el apoyo suficiente para no decaer cuando todo parecía complicado e imposible.

Asimismo, quisiera agradecer a todos mis amigos por brindarme el apoyo y diversión que necesito fuera de las aulas.

En segundo lugar, me gustaría agradecer a Iñigo Fermín Ezcurdia, director de este trabajo, por el esfuerzo y la ayuda que me ha proporcionado en la redacción de este documento. Sin su dedicación, la calidad de este documento hubiese sido notablemente menor. Gracias por darme la oportunidad de realizar este trabajo, por tus consejos, correcciones y por atender a todas mis preguntas e inquietudes.

A la Universidad Pública de Navarra por darme la oportunidad de realizar mis estudios y en especial a todos los profesores que han tomado parte en mi formación y que en mayor o menor medida han contribuido en ella.

Y por último, agradecer a Javier Jorge Soteras, codirector del trabajo, por darme la oportunidad de realizar las prácticas de este trabajo en la empresa Hiberus Tecnología.

A todos ellos muchísimas gracias.

Resumen

Este proyecto recorre las fases de diseño, implementación y evaluación de una herramienta web de apoyo para una empresa dedicada al desarrollo web. Esta herramienta recibe como entrada un formulario HTML básico, analiza automáticamente el código provisto y ofrece una interfaz web mediante la que se puede configurar un sistema de reglas y relaciones entre inputs que especifican qué elementos deben ser mostrados/ocultados y bajo qué condiciones. Teniendo como resultado un código JavaScript que reúna todas las reglas configuradas. Finalmente, se han realizado pruebas de usuario para validar su utilidad y usabilidad.

Este proyecto, se ha llevado a cabo bajo la supervisión de la empresa Hiberus Tecnología.

Palabras clave

Conviene definir inicialmente algunas palabras clave que van a ser utilizadas a lo largo de la presente memoria. Estas palabras son:

- **JS**: Abreviatura del lenguaje JavaScript. Lenguaje de programación o de secuencias de comandos que permite implementar funciones e interacciones complejas en páginas web.
- **Input:** Conjunto de datos que se introducen en un sistema o un programa informático en esta herramienta se hará referencia a las etiquetas input, select y option.
- **Input bloque**: Conjunto de datos que se introducen en un sistema o un programa informático que harán referencia al resto de etiquetas, div, h2 entre otros.
- **HTML:** HyperText Markup Language. Código que se utiliza para estructurar y desplegar una página web y sus contenidos
- **NASA-TLX:** herramienta de evaluación subjetiva, multidimensional y ampliamente utilizada que califica la carga de trabajo percibida para evaluar la efectividad de una tarea, sistema o equipo u otros aspectos del desempeño.
- **CSS:** Abreviatura de Cascading Style Sheets lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado.
- **DB:** Abreviatura de Data Base, es decir, base de datos.
- **Plantilla:** Es un medio o aparato o sistema, que permite guiar, portar, o construir, un diseño o esquema predefinido. Proporciona una separación entre la forma o estructura y el contenido.
- **Formulario:** Elemento de una interfaz web donde se agregan los diferentes campos de entrada de datos o de confirmación, así como los botones, que como mínimo ha de haber uno, el de envío.
- **Regla:** Norma establecida para el formulario. Es decir, principio que se impondrá para dirigir la conducta o la correcta realización de una acción o el

correcto desarrollo de una actividad dentro del formulario. Se utilizará para definir qué input se ha de mostrar/ocultar.

- **Scraping**: técnica utilizada mediante programas de software para extraer información de sitios web. Estos programas normalmente imitan la forma que tiene un usuario de navegar en la red y recopilan la información indicada en el algoritmo desarrollado.
- **Controlador**: Parte de una aplicación que hace referencia a lo que controla las funciones de una aplicación.

Contenido

Introducción

Problema a resolver

La propuesta de este proyecto la transmitió la empresa Hiberus Tecnología, con la cual me puse en contacto para realizar prácticas.

La idea del proyecto surgió de la necesidad de contar con una herramienta implementada en NodeJs que facilitase la generación de un código JavaScript reuniendo todas las reglas configuradas que ayude a la adición de inteligencia y secuencialidad de formularios.

Por otro lado, se quiso que la herramienta a implementar tuviera algunas características concretas como:

- Mantener una arquitectura MVC (Modelo Vista Controlador).
- Utilizar Node.js como servidor web
- Emplear Edge como motor de plantillas para generar el frontal de la aplicación.
- Utilizar Bootstrap para el diseño de la aplicación.
- Manejar la herramienta en varios idiomas, mínimo dos.

La intención es realizar una herramienta de uso intuitivo, se desea facilitar el trabajo a aquellos quienes deban utilizar la herramienta asiduamente, a pesar de estar orientada a desarrolladores profesionales.

Hiberus Tecnología

Hiberus Tecnología es la empresa en la que realicé prácticas de empresa además de realizar el proyecto con ellos. Hiberus Tecnología es una de las principales compañías de tecnología españolas la cual están especializados en servicios de consultoría de negocio, desarrollo tecnológico, transformación digital y outsourcing además, prestan servicios a organizaciones públicas y privadas a las que ayudamos a satisfacer sus necesidades de negocio.

Este proyecto surgió de la necesidad de tener una herramienta implementada en NodeJs para realizar reglas teniendo como resultado un código JavaScript que reúna todas las reglas configuradas. Actualmente, ellos utilizan una plataforma, Liferay, la cual contiene un componente para crear formularios y además configurar un sistema

de reglas y relaciones entre inputs que especifican qué elementos deben ser mostrados/ocultados y bajo qué condiciones.

Liferay, es una plataforma que se apoya en la tecnología JAVA y que permite crear comunidades enteras, blogs… con tan solo arrastrar un portlet, componente modular de las interfaces de usuario que se gestionan y visualizan en un portal web.

Pero en el momento que se creaba un formulario por otro medio que no fuera usando la plataforma Liferay no se disponía de ninguna herramienta para obtener el código JavaScript de las reglas que se deseaban generar.

Por lo tanto, obtener el código JavaScript que reúna todas las reglas configuradas supone un avance importante para la empresa ya que dispondrán de otro método para gestionar reglas en los formularios.

Objetivos de Proyecto

Tras una reunión con el equipo de Hiberus se definieron los siguientes objetivos de proyecto:

- Diseñar e implementar una aplicación en formato servicio web que no requiera de instalación de software adicional para el usuario final.
- El servicio debe ser capaz de extraer meta información sobre los formularios y sus inputs de manera automatizada.
- Estos formularios podrán aportarse mediante un copiado y pegado de bloques HTML o indicando una URL a ser scrapeada para la obtención de los formularios contenidos en esta.
- En el caso de hallar dos o más formularios, el usuario debe poder escoger sobre cuál de ellos trabajar.
- Configurar un sistema de reglas en base a un formulario HTML básico permitiendo al usuario diseñar cuantas reglas desee sobre cualquiera de los inputs del formulario.
- Las reglas deben poder aplicarse sobre los tipos de input más comúnmente utilizados en formularios HTML. Ejemplo: text, radio, checkbox...
- Se ofrecerá un abanico amplio de posibilidades en los filtros y operaciones a aplicar. (Ejemplo: igual, entre, contiene, no contiene…)
- El servicio debe generar código Javascript que reúna las reglas configuradas mediante su interfaz.

- El código Javascript debe aplicarse automáticamente sobre los formularios originales, sin necesidad de readaptar o modificar dichos formularios.
- El código Javascript debe poder ser exportado como fichero o como bloque de texto.
- Implementar un sistema de persistencia de datos basado en usuarios que permita almacenar y personalizar los menús y utilidades ofrecidas según el usuario identificado.
- El servicio debe ofrecer la posibilidad de modificar reglas y formularios creados con anterioridad.
- Ofrecer un entorno multilenguaje que establezca las bases para el posterior añadido de nuevos idiomas.
- Auditar la usabilidad y conveniencia del servicio.

Objetivos Personales

Durante la implementación de esta herramienta, he tenido en cuenta diferentes objetivos personales.

- Aplicar una metodología de análisis, diseño e implementación de software avalada e integrada en el entorno profesional de Hiberus Tecnologías.
- Aprender a desarrollar proyectos con el entorno NodeJs
- Profundizar mis conocimientos de programación de los siguientes lenguajes:
    - JavaScript
    - HTML
    - CSS/Bootstrap
- Aprender a elaborar y administrar bases de datos con MongoDB.
- Aprender a utilizar plantillas Edge.
- Aprender a trabajar en un entorno de trabajo profesional.
- Satisfacer las necesidades de la empresa, y a la vez, satisfacerme a mí misma.
- Poner en práctica las competencias adquiridas durante el grado.

Planificación

Para la planificación del proyecto se ha realizado una división del proyecto en las distintas etapas o tareas a realizar.

Estas etapas junto con sus tareas son:

- Requisitos
    - Búsqueda de documentación: En esta tarea se realizará la búsqueda de la documentación necesaria para comenzar con el proyecto. Es decir, recopilar información sobre el actual funcionamiento de la empresa, sobre el estado del arte de herramientas similares y sobre las tecnologías y herramientas establecidas como requisitos para la implementación del servicio.
    - Introducción y planificación.
    - Elicitación de requisitos: reuniones con el codirector para la recogida de características que debe tener la herramienta.
- Diseño e implementación
- Verificación: Testear la herramienta en busca de posibles fallos.
- Documentación: Generación de la documentación necesaria

Las tareas de la primera etapa, carecen de exactitud, ya que a la hora de realizar el proyecto se han ido añadiendo nuevos requisitos, no siendo así para el resto de tareas que sí expresan con mayor precisión la información real.

A continuación, se muestra el diagrama de Gantt que recoge toda la información pertinente a la división comentada con anterioridad junto con sus estimaciones. Se ha de tener en cuenta que para la elaboración de la tabla se ha considerado que solo hay una persona realizando el proyecto.

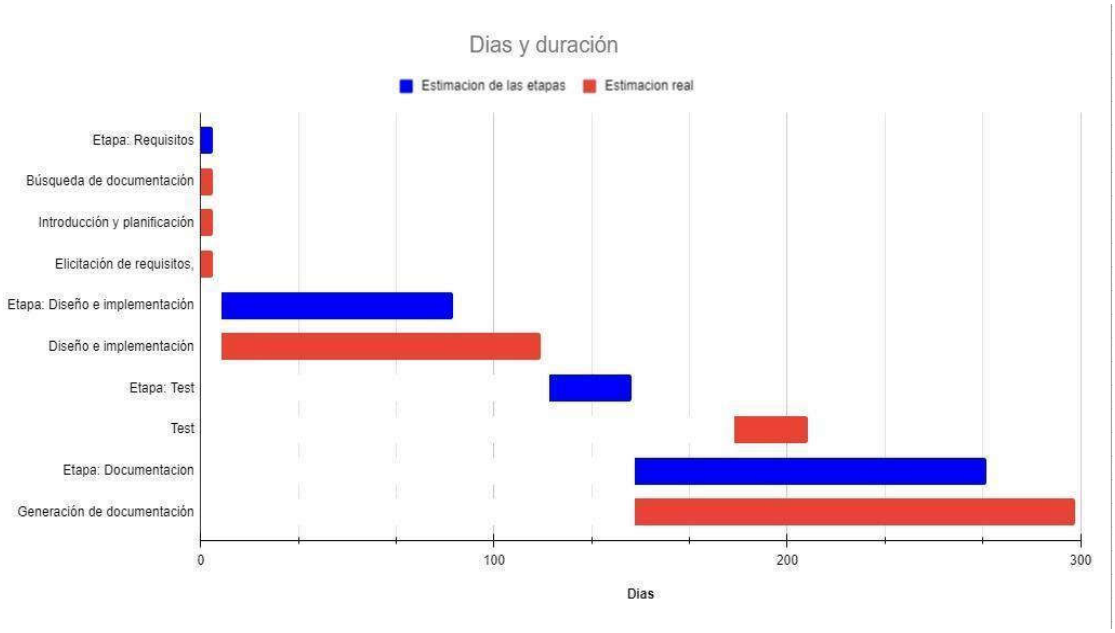


Ilustración 1 - Diagrama de Gantt

Se ha de mencionar que las estimaciones se han basado en los días necesarios para realizar las tareas, teniendo en cuenta que en un día se ha trabajado de media 5 horas. Por otro lado, la estimación inicial de las etapas está representada en la barra azul mientras que la estimación real de cada tarea es la barra roja.

Finalmente, la gráfica de Gantt muestra que ha habido un retraso en realizar algunas tareas, esto se ha debido a que inicialmente se hizo una estimación con los requisitos iníciales pero durante la etapa de diseño e implementación se añadieron algunos de los requisitos por lo que se necesitó más tiempo para realizar la tarea.

Metodología empleada

Dentro de la Ingeniería del software existen varias metodologías para llegar a la construcción final de un producto de software y optimizar el desarrollo del mismo. Cada metodología tiene ventajas y desventajas de tal forma que para realizar un buen proyecto, en cada proyecto se tiene que elegir el que mejor se adapte a sus necesidades, tanto para la creación, elaboración y mantenimiento.

Una metodología es un conjunto de técnicas las cuales te permiten realizar correctamente cada una de las fases del ciclo de vida de tu proyecto.

Después de hacer un estudio previo de cómo iba a ser mi aplicación, y qué pasos se iban a seguir para implementarla, se optó por utilizar la metodología basada en cascada.

Se decidió utilizar esta metodología porque sigue una serie de etapas de forma sucesiva, la etapa siguiente empieza cuando termina la etapa anterior, además porque resulta más cómodo para proyectos realizados por una sola persona ya que está obligado a contemplar cada paso del desarrollo, pudiendo demostrar los distintos tipos de competencias adquiridas durante la carrera.

Las fases que componen el modelo son las siguientes:

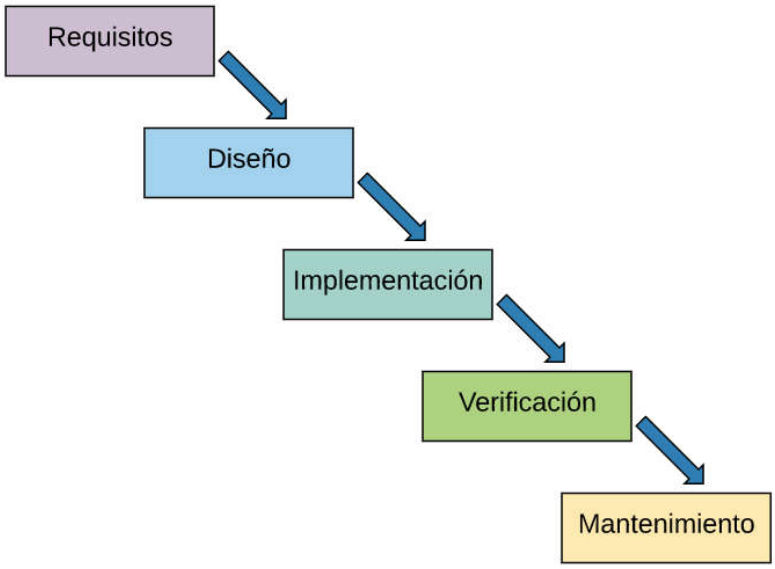


Ilustración 2 – Fases de la metodología basada en cascada

Se ha de mencionar que las etapas Búsqueda de documentación, Introducción y planificación, Elicitación de requisitos creadas en la planificación del proyecto pertenecen a la fase Requisitos ya que esta fase es la encargada analizar las necesidades del cliente para determinar las características del software a desarrollar, y se especifica todo lo que debe hacer el sistema sin entrar en detalles técnicos.

Por otro lado, cabe destacar que se han realizado reuniones semanales en la empresa mientras que con mi director se ha realizado reuniones mensuales para verificar el avance del proyecto.

Requisitos funcionales

En esta sección se pretende establecer los requisitos funcionales de la aplicación, mediante casos de uso.

Diagramas de Caso de Uso

Los diagramas de casos de uso describen en forma de acciones y reacciones el comportamiento de un sistema desde el punto de vista del usuario, ya que permiten definir los límites y las relaciones entre este y el entorno.

Además, son descripciones de la funcionalidad del sistema independientes de cómo se realice la implementación.

- Usuarios no registrados, serán aquellos que interactúan por primera vez con la herramienta y no dispongan de una cuenta para iniciar sesión.
- Usuarios registrados, aquellos que ya han interactuado con la herramienta y disponen de una cuenta de usuario.

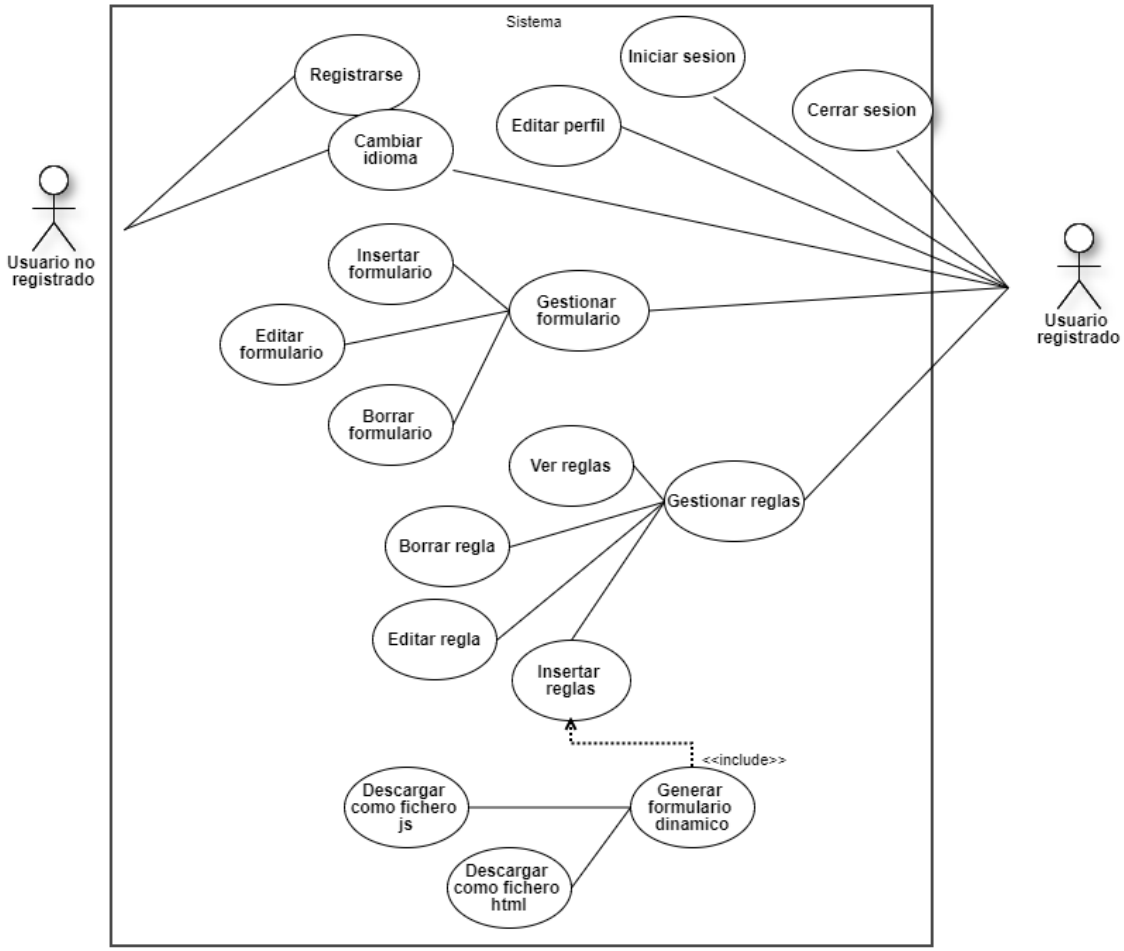


Ilustración 3 - Diagrama de casos de uso

Diagrama de secuencia

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso.

Lo interesante de esta herramienta es devolver código JavaScript con las reglas generadas por lo tanto, únicamente un usuario registrado es capaz de lograr este objetivo y para ello mediante el siguiente diagrama de secuencia se mostrara los pasos que ha de seguir un usuario registrado para lograr el código JavaScript con las reglas generadas.

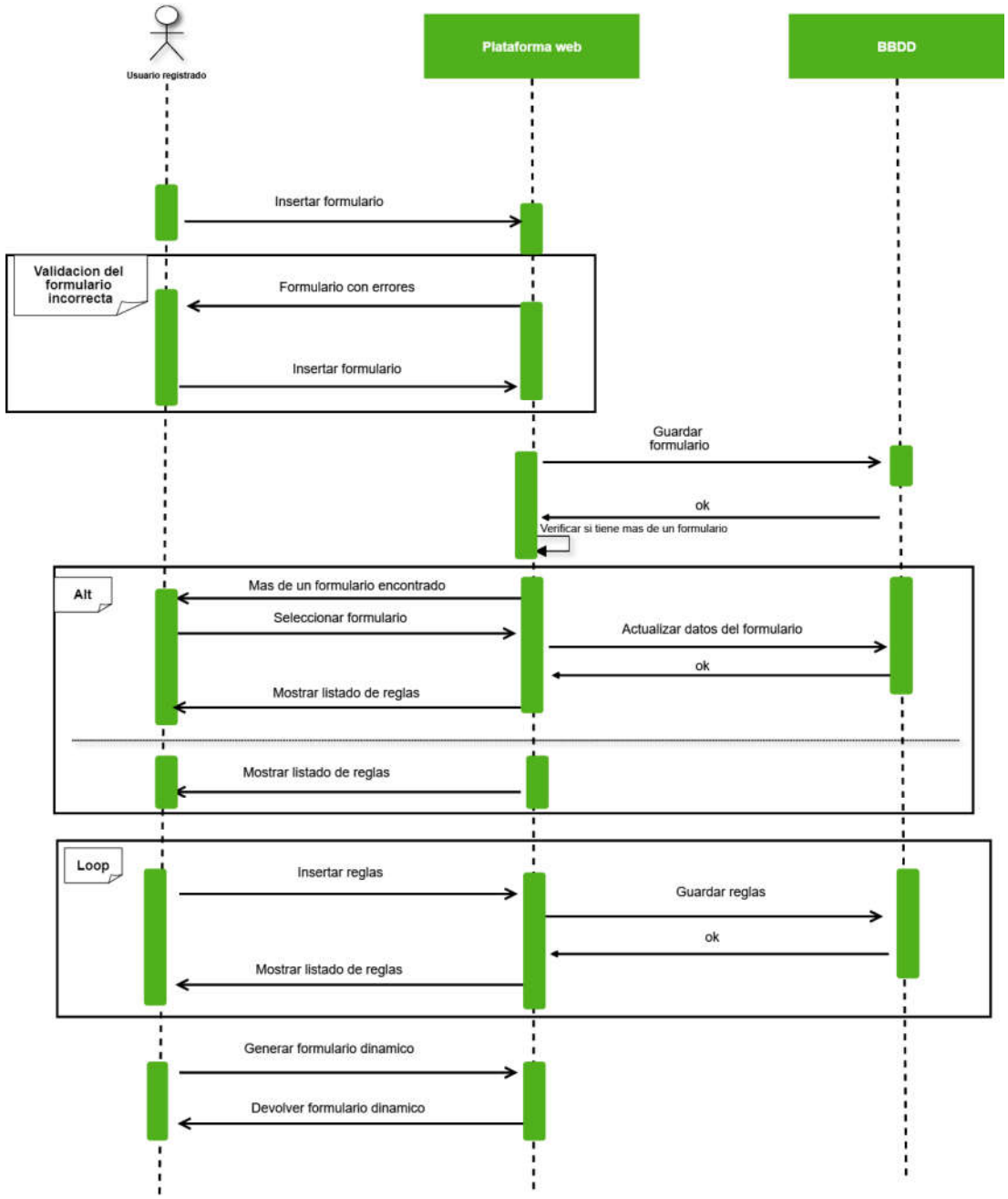


Ilustración 4 - Diagrama de secuencia

Diagrama de flujo

Es interesante fijarse en el flujo de la generación de un formulario dinámico ya que implica inserciones y modificaciones en la base de datos tanto en su creación como cuando cumpla su función.

Para generar un formulario dinámico únicamente los usuarios registrados tienen acceso. Una vez iniciado sesión en la plataforma, lo interesante de esta herramienta es poder generar formularios dinámicos. El diagrama de flujo principal de la aplicación que modela este proceso se expone en la siguiente figura.

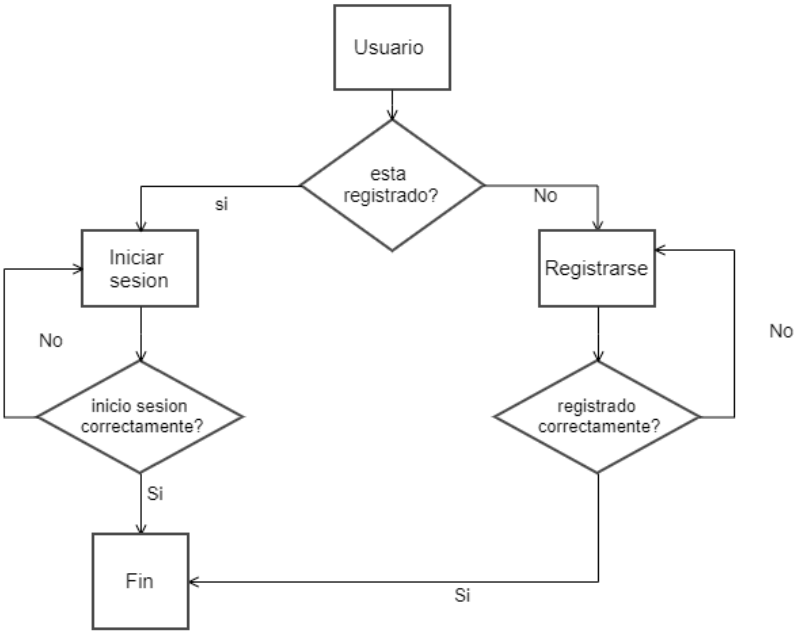


Ilustración 5 - Diagrama de flujo iniciar sesión

Una vez iniciado sesión en la plataforma, lo interesante de esta herramienta es poder generar formularios dinámicos. El diagrama de flujo principal de la aplicación que modela este proceso se expone en la siguiente figura.

Hay que tener en cuenta que cada vez que se realiza una validación incorrecta se mostrará un mensaje de error distinguiendo el tipo de error dado.

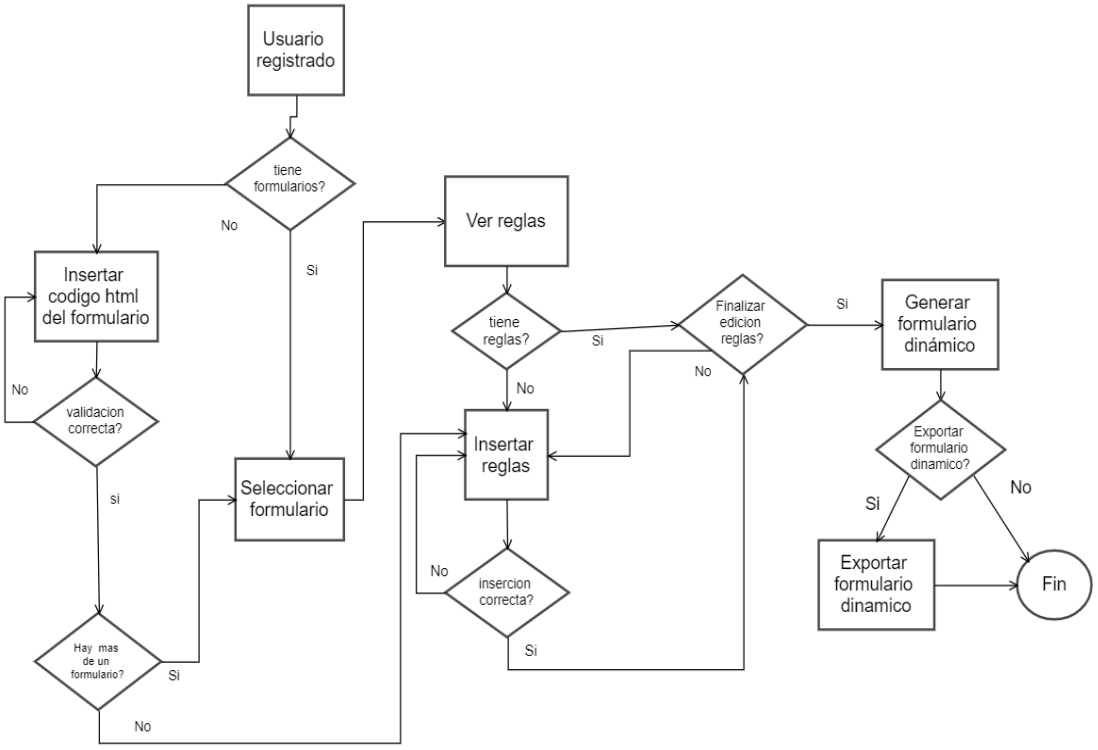


Ilustración 6 - Diagrama de flujo generar formulario dinamico

Modelo de datos

Se procede ahora al diseño de la capa de persistencia de datos del sistema. Se necesita un modelo de datos que represente todos los datos que se almacenan en la plataforma web, independientemente del sistema de gestión de base de datos que se emplee finalmente.

Usaremos Diagramas Entidad-Relación (DER) para representar nuestra Base de Datos fina.

Diagrama entidad-relación

Un modelo entidad-relación es una herramienta para el modelo de datos, la cual facilita la representación de entidades de una base de datos, es decir, nos muestra los datos relevantes del sistema así como las relaciones entre esos datos.

Los datos relevantes del sistema se representan mediante Entidades, las cuales se relacionan mediante una serie de asociaciones que definen una serie de información relevante para el sistema.

Teniendo en cuenta lo anterior, se podría obtener un diagrama de entidad-relación para nuestro sistema con las siguientes características:

- Para cada entidad, se indicará la clave primaria de la tabla final que representará dicha entidad.
- Para cada relación, se expresará su cardinalidad mediante la notación X:Y, donde X es la multiplicidad mínima e Y es la máxima de cada entidad que participe en la relación.

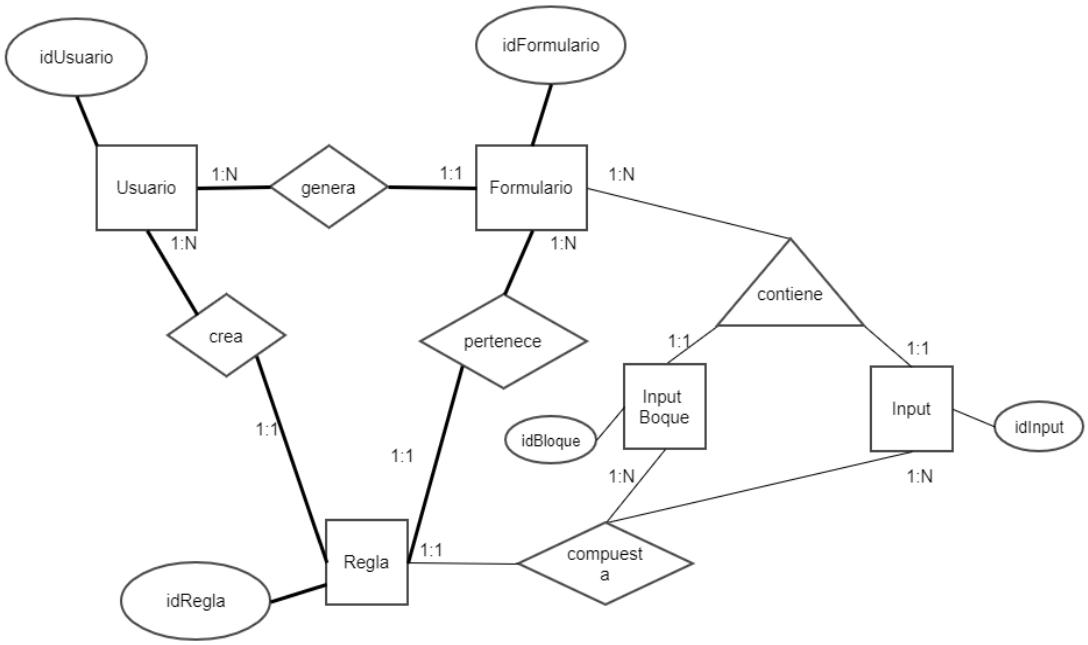


Ilustración 7 - Diagrama entidad - relación

Relaciones del modelo de datos

A continuación se muestra una visión más detallada de las relaciones entre las entidades del sistema.

Genera		
Entidades	**Cardinalidad**	**Descripción**
Usuario	1:N	Los usuarios del sistema pueden generar más de un formulario mientras que un formulario únicamente pertenece a un usuario.
Formulario	1:1	

Pertenece

Entidades	Cardinalidad	Descripción
Formulario	1:N	Un formulario puede contener más de una regla mientras que una regla únicamente pertenece a un formulario.
Regla	1:1	

Crea		
Entidades	**Cardinalidad**	**Descripción**
Usuario	1:N	Un usuario del sistema puede crear más de una regla mientras que una regla únicamente pertenece a un usuario.
Regla	1:1	

Compuesta		
Entidades	**Cardinalidad**	**Descripción**
Regla	1:2	Una regla puede estar compuesta por input o input bloque, siendo la combinación de ellas es decir, una regla puede estar compuesta por un input y un input bloque o por dos input o por dos input bloque. En cambio, un input o un input bloque puede aparecer en varias reglas.
Input	1:N	
Input bloque	1:N	

Contiene		
Entidades	**Cardinalidad**	**Descripción**
Formulario	1:N	Un formulario puede contener más de un input e input bloque, en cambio, un input y un input bloque únicamente pertenecen a un formulario
Input	1:1	
Input bloque	1:1	

Herramientas y Tecnologías empleadas

JavaScript

JavaScript es un lenguaje de programación interpretado, muchas veces nos lo vamos a encontrar escrito de la siguiente forma 'JS'. Se define como orientado a objetos, basado en prototipos.

Se utiliza principalmente para realizar aplicaciones web, donde normalmente es aplicado al lado del cliente. De esta forma permite añadir mejoras en la interacción y funcionalidades complejas y lógica de negocio a ser ejecutada por parte del usuario, realizando los sitios más atractivos y llamando la atención del usuario.

Como se ha mencionado, JS, es un lenguaje interpretado, esto quiere decir que no es necesario compilar el código para poder ejecutar la aplicación. Una de sus características más importantes es que se ve en todo momento lo que se está haciendo, es decir, una vez realizado un cambio en tu código, lo ves al instante.

Por lo tanto, JavaScript es el lenguaje de programación utilizado en la herramienta.

NodeJs

Nodejs es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación JavaScript. Esta herramienta fue creada con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web.

¿Por qué resulta tan atractivo entonces? Supongamos la entrada de dos peticiones a la vez al backend, lo habitual es realizar una petición y una vez finalizada esta realizar la siguiente, NodeJS es capaz de realizar ambas a la vez proporcionando la salida de los datos independientemente la una de la otra, reduciendo los tiempos en las peticiones entre otras cosas.

Una de las ventajas que ofrece NodeJS frente a otros lenguajes usados en el lado del servidor, como por ejemplo Java o PHP. Cualquiera de estos dos últimos lenguajes, al recibir una petición, crea un nuevo hilo para responder a la misma. Por otro lado, NodeJS, en vez de crear un nuevo hilo de ejecución, lo que hace es lanzar un evento dentro del proceso que está atendiendo la petición. De esta forma, se reduce el coste

computacional y de recursos del sistema permitiendo mantener más conexiones simultáneas con los mismos recursos.

Otra de las ventajas que ofrece, es que su arquitectura es no bloqueante, por lo que se puede realizar operaciones I/O de forma asíncrona. Esto permite realizar tareas de forma simultánea en un único proceso. De hecho, estas son las características que permiten hacer que los sistemas desarrollados en NodeJS sean escalables, aunque esto dificulta en algunos casos el diseño de las aplicaciones.

Módulos node utilizados:

- Axios: versión utilizada 0.20.0. Axios facilita el envío de solicitudes HTTP asíncronas a puntos finales REST y realiza operaciones CRUD.
- Body-parser: versión utilizada 1.19.0. Responsable de analizar los cuerpos de las solicitudes entrantes en un middleware antes de manejarlo.
- Express: versión 4.17.1. Es un marco de aplicación web de back-end para Node.js usado para proporcionar lógica del lado del servidor para aplicaciones web y móviles.
- Express-edge: versión utilizada 2.0.2. Utiliza el motor de plantillas Edge con Express, donde el motor de plantillas Edge, permite separar los elementos html que se repiten en cada una de las secciones como el encabezado, menú y pie de página. Estos se guardarán en una archivo y las demás secciones se extenderán de este archivo principal y de esa manera se logra un código más organizado.
- Express-session: versión utilizada 1.17.1. Es un middleware de sesión encargado de manejar el proceso de creación y destrucción de sesiones y su gestión de cookies.
- html-validator: versión utilizada 5.1.14 es un programa de garantía de calidad que se utiliza para comprobar los elementos de marcado del lenguaje de marcado de hipertexto (HTML) en busca de errores de sintaxis.
- i18n-express : versión utilizada 0.11.1 es el proceso de diseño y preparación de productos de software (aplicaciones) para admitir múltiples configuraciones regionales, idiomas y regiones
- jquery versión utilizada 3.5.1. simplifica la tarea de programar en JavaScript y permite agregar interactividad a un sitio web.
- jsdom: versión utilizada 16.3.0. permite recrear un elemento DOM dentro de un entorno en el que no contamos con un navegador. DOM, abreviatura de Document Objet Model, sus objetos modelizan tanto la ventana del navegador

como el historial, el documento o página web, y todos los elementos que pueda tener dentro la propia página, como párrafos, divisiones, tablas, formularios y sus campos, etc.

- mongoose: versión utilizada 5.9.22. Gestiona las relaciones entre los datos, proporciona validación de esquemas y se utiliza para traducir entre objetos en código y la representación de esos objetos en MongoDB.

Estas dependencias se declaran en un fichero llamado “packaje.json” y se descargan en el proyecto llamando simplemente al comando “npm install”. Una vez descargadas se pueden referenciar mediante el sistema de importación de módulos que proporciona NodeJS.

Ilustración 8 - NodeJs

Bootstrap

Se trata de una multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web que contiene plantillas de diseño con tipografías, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basados en HTML y CSS, así como extensiones de JavaScript adicionales.

Bootstrap está orientado a facilitar el diseño y elaboración de aplicaciones web utilizando un sistema de cuadrículas que ayuda a conseguir interfaces claras, haciendo que las aplicaciones se vean con diferentes diseños en diferentes dispositivos, dependiendo del espacio del que disponga la aplicación (ordenadores personales, tablets, móviles, etc.), por lo que el programador sólo tiene que preocuparse de utilizar las clases css correctamente, olvidándose prácticamente de los quebraderos de cabeza que pueden proporcionar las hojas de estilo.

En este proyecto se ha utilizado BootStrap para dotar al frontend de la aplicación web de un look&feel común entre sus diferentes secciones y de una estética que presente características de usabilidad avaladas por la comunidad.

Ilustración 9 - Bootstrap

MongoDB

MongoDB es la base de datos utilizada en esta herramienta.

MongoDB es una base de datos de documentos que ofrece una gran escalabilidad y flexibilidad, y un modelo de consultas e indexación avanzado que está orientada a documentos. Permitiendo que algunos de estos campos sean listas y objetos embebidos, dando aún más versatilidad y posibilidades a la creación de la estructura de datos.

La nomenclatura respecto a las bases de datos SQL, cambia. Lo que en una base de datos SQL era una tabla, en MongoDB es una colección, y lo que sería una fila en SQL se corresponde con un documento. Así pues, una base de datos MongoDB está compuesta por colecciones, que a su vez están compuestas por documentos.

Una de las ventajas de usar MongoDB, es que la consola de la base de datos puede ejecutar JavaScript. Esto permite escribir scripts en este lenguaje para realizar consultas complejas, o tareas de mantenimiento del servidor. Además, mediante el uso de JavaScript permite definir funciones, que pueden ser llamadas posteriormente ahorrando tiempo y líneas de código.

A la hora de realizar consultas, MongoDB tiene su propio sistema de querys. Utilizando este sistema permite realizar búsquedas por campos, rangos, o incluso realizar búsquedas utilizando expresiones regulares. Además, permite definir índices de búsqueda propios, permitiendo así aumentar el rendimiento notablemente en consultas que se vayan a repetir frecuentemente.

Ilustración 10 - MongodDB

Robo 3T

Es una interfaz gráfica de usuario (GUI) de escritorio popular para sus implementaciones de alojamiento MongoDB que le permite interactuar con sus datos a través de indicadores visuales en lugar de una interfaz basada en texto. Esta herramienta de código abierto tiene soporte multiplataforma y en realidad integra el shell mongo dentro de su interfaz para proporcionar interacción basada tanto en shell como en GUI.

Por lo tanto, Robo 3T es la interfaz gráfica utilizada para interactuar con los datos de MongoDB.

Ilustración 11 - Robo 3T

Base de datos

Según los requisitos proporcionados se ha diseñado una sencilla base de datos, utilizando MongoDB, que satisface las necesidades y exigencias del cliente. En este

caso existen varias restricciones que se nos imponen y que dan lugar a varias tablas y relaciones que no se han contemplado con los requisitos recopilados.

Para conectar la base de datos se necesita el módulo “mongoose“ ofrecido por node.js. Mongoose es un driver de MongoDB para NodeJS el cual permite definir el esquema de los documentos de las colecciones como objetos. Además permite añadir métodos a estos objetos, facilitando así la separación de las interacciones con la base de datos del resto del código.

Para realizar la conexión se realiza mediante las instrucciones en el controlador, fichero index.js:

```
const mongoose = require('mongoose');
mongoose.connect(`mongodb://localhost/SmartForms`);
```

Donde *SmartForms* es el nombre de la base de datos almacenada en MongoDB.

Por otro lado, en el proyecto se ha de tener una sección donde se creen los modelos necesarios, es decir, las tablas que tendrá la base de datos.

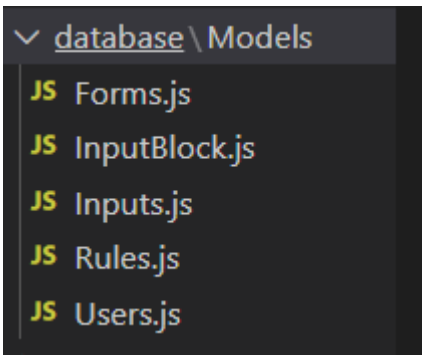


Ilustración 12 – Sistema de ficheros, carpeta database/Models

Por lo que manualmente se tendrá que definir los atributos que tendrán dichas tablas, a continuación se muestra un ejemplo del esquema Users:

```
const mongoose = require('mongoose');

const UsersSchema = new mongoose.Schema({
      userName:String,
      firstSurname:String,
      secondSurname:String,
      email: String,
      pass: String,
```

```
      phone:Number
});
const Users = mongoose.model('Users', UsersSchema);
module.exports = Users;
```

Implementación de servicio

Sistema de ficheros

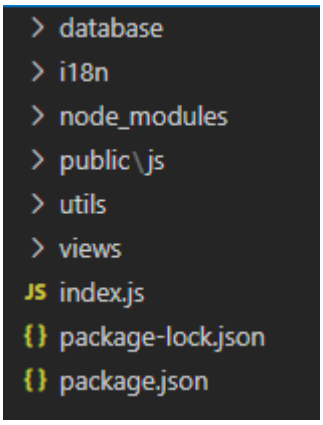


Ilustración 13 - Sistema de ficheros

node_modules

En este directorio se encuentran todos los módulos instalados mediante npm. Al añadir un nuevo módulo al fichero “package.json” y ejecutar “npm install” se crea esta carpeta con todos los módulos declarados como dependencias del proyecto.

public

Esta carpeta es de acceso público, por lo que todos los archivos que se coloquen en ella serán accesibles a través de peticiones http al servidor. En esta carpeta se colocarán las hojas de estilos CSS, el JavaScript que se va a ejecutar en la parte del cliente, y las imágenes del proyecto.

En este proyecto se ha utilizado CSS ya que los estilos se han aplicado mediante Bootstrap por lo tanto únicamente habrá una carpeta JS con los archivos js necesarios.

Si se deseara aplicar estilos mediante CSS, habría que crear una carpeta dentro de public con el nombre CSS y dentro de este almacenar los ficheros css necesarios, como se muestra a continuación:

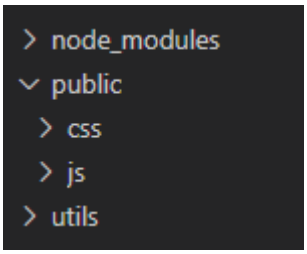


Ilustración 14 - Sistema de ficheros, carpeta public

utils

Este es el directorio en el que depositamos todos los ficheros JavaScript encargados de tratar las peticiones HTTP. Es esta carpeta se han generado dos ficheros: “Events.js” y “Procesor.js”

El primero tendrá las funciones necesarias para generar el javaScript con las reglas configuradas así como para generar y obtener dicho snippet javascript al usuario. Se analiza el fichero “Events.js” con mayor profundidad en la sección Generar javaScript con las reglas almacenadas.

Mientras que el segundo fichero, “Procesor.js” será el encargado de obtener toda la información relacionada de un input, es decir, se obtendrán los id, name, value, class y type de los input y asimismo, el querySelector que le corresponde. La manera en la que se obtiene dicha información se analiza en profundidad en la sección Obtención de input y sus atributos.

QuerySelector permite acceder a un elemento por su valor de atributo id, o a la colección de elementos cuya etiqueta es de un determinado tipo o tiene como atributo name un valor concreto.

Es decir, dado el siguiente formulario el querySelector del input seria: *form:nth-child(1) > div:nth-child(1) > input:nth-child(1)*

```
<form>
        <div >
                <input type="text" id="ap1" name="ap1">
        </div>
</form>
```

A su vez, este fichero se encargará de obtener los querySelector por bloques, es decir, aquellas etiquetas que no sean meta, title, input ,textarea , select, option, br.

views

En esta carpeta se almacenan los ficheros que el sistema de templating usa para "renderizar" la página.

Las plantillas creadas son las siguientes:

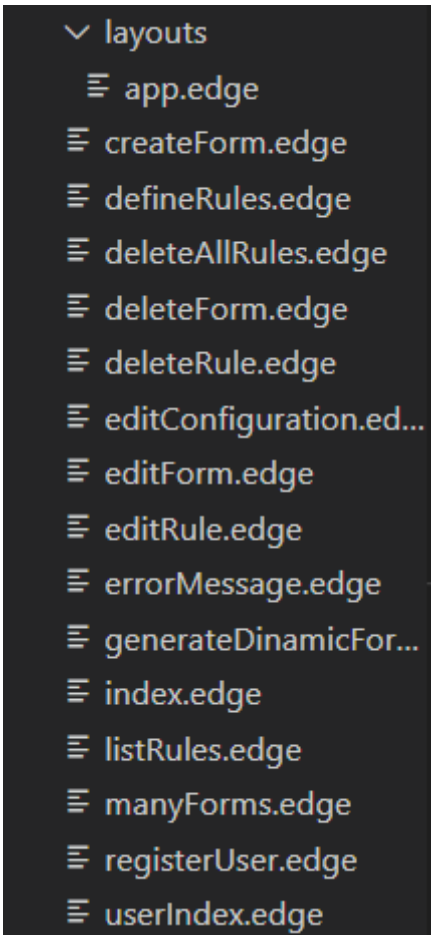


Ilustración 15 - Sistema de ficheros, carpeta views

Donde la carpeta layouts contendrá la plantilla Edge principal 'app.edge'. El uso de plantillas '.edge' queda explicado en la sección Utilización de las plantillas edge.

Cada plantilla es una vista específica de la herramienta, donde index.edge será la vista inicial, es decir, si se escribe en el navegador la URL de nuestro servidor, éste nos responderá con la página de inicio de la aplicación. En este entorno de preproducción la URL sería http://localhost:4000/

La plantilla userIndex.edge será la vista a la que se accedera una vez se haya iniciado sesión en la herramienta. Respecto al resto de plantillas mostrarán las vistas como su nombre indica es decir, la plantilla createForm.edge mostrará la vista para insertar un formulario.

Edge nos permite definir de forma sencilla el HTML de la página y la inclusión de contenido dinámico. Esto se consigue proporcionando como argumentos parámetros que luego se usarán al "renderizar" la página. Esto se realiza mediante la llamada a la función:

```
var user = await Users.findById(users._id);

res.render('editConfiguration',{
        lang,
        user
});
```

En esa llamada, le estamos pasando como argumento al render el nombre del fichero edge que se quiere visualizar además de poder pasar parámetros. Para hacer referencia a dichos parámetros en las plantillas edge se deberá de realizar mediante la utilización de *{{nombre del parámetro}}*, a continuación se muestra un ejemplo del modo de hacer referencia a dichos parámetros.

```
<input id="your_username" name="your_username" type="text"
placeholder="{{user.userName}}" class="form-control">
```

También se puede observar que user es un usuario almacenado en la base de datos por lo que, en la plantilla también se podrá hacer referencia a un campo específico de la base de datos, en este caso al *userName* del usuario el cual hace referencia al nombre del usuario.

index.js

En este fichero se realiza la mayor parte de la configuración de la aplicación. En él se define el sistema de templating, cómo tratar algunos errores, y definir las distintas vistas que utiliza la aplicación.

Para realizar la configuración del puerto del servidor se hará mediante las siguientes instrucciones, utilizando el puerto 4000 durante el desarrollo de la herramienta:

```
app.set('port',4000);


app.listen(app.get('port'), ()=> {
        console.log('Server on port',app.get('port'));
});
```

package.json

El fichero “package.json” contiene información general de la aplicación, como por ejemplo, nombre, versión, etc... Así como la información relativa a las dependencias de la aplicación.

Utilización de las plantillas edge

A lo largo de la memoria se ha mencionado varias veces el uso de plantillas Edge en la herramienta.

Dichas plantillas Edge nos permite definir de forma sencilla el HTML de la página, y además de permitir la inclusión de contenido dinámico además de permitir separar los elementos html que se repiten en cada una de las secciones como el encabezado, menú y pie de página.

Para poder utilizar las plantillas Edge en la herramienta inicialmente se ha de importar la librería Express-edge mediante la instrucción npm install express-edge. En segundo lugar, en el fichero de configuración index.js se ha de incluir las siguientes instrucciones:

```
const {config, engine} = require('express-edge');
app.use(engine);
app.set('views',`${__dirname}/views`);
```

Estas instrucciones indicarán el uso de la librería express-edge y el directorio en el que se encuentran las plantillas que se usarán en la herramienta.

Una vez realizada la configuración, se necesitará un fichero el cual define la estructura de la plantilla, dicho fichero se llamará app.edge y se encontrará en views/layouts/app.edge se tendrá en la carpeta layouts para diferenciarlo del resto de archivos en la carpeta views.

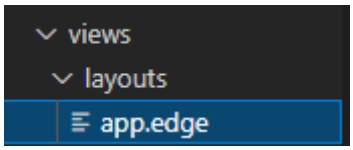


Ilustración 16 - Plantilla base

A continuación se mostrará el código del archivo app.edge, archivo base de la plantilla, estas secciones se representan como @!(nombre de la sección) por lo tanto, se puede observar que se tiene tres secciones, sección Title (representará el título de cada vista), sección translation (representa la opción de multi idioma) y la sección Content (el cual contendrá el contenido de la vista).

```
<!DOCTYPE html>
<html>
	<head>
		<title>Form</title>
		<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
integrity="sha384-
MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous">
		<script src="https://code.jquery.com/jquery.js"></script>
		<script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js"></script>
		<script src='js/Functions.js' type='text/javascript'></script>
		<!--Para el buscador en el selector-->
		<link href="https://cdn.jsdelivr.net/npm/select2@4.1.0-
beta.1/dist/css/select2.min.css" rel="stylesheet" />
		<script src="https://cdn.jsdelivr.net/npm/select2@4.1.0-
beta.1/dist/js/select2.min.js"></script>
		<!--Para el buscador de formularios-->
		<link href="https://cdn.datatables.net/1.10.16/css/jquery.dataTables.min.css"
rel="stylesheet"/>
		<script
src="https://cdn.datatables.net/1.10.16/js/jquery.dataTables.min.js"></script>
	</head>
	<body onload="myFunction()">
		<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
			<div class="container">
				@!section('title')
			</div>
			<div>
				@!section('translation')

			</div>
		</nav>
		@!section('content')
	</body>
</html>
```

Por otro lado, los archivos que extienden de app.edge, han de tener una referencia a dicho fichero app.edge, es decir, si se genera una nueva plantilla edge, supongamos

que la nueva plantilla se llama ejemplo.edge, esta nueva plantilla tiene que tener una referencia a app.edge y dicha referencia es la siguiente:

```
@layout('layouts.app')
```

Esta instrucción expresa que se extiende del fichero app situado en la ruta layouts.

Finalmente, para definir las secciones en las nuevas plantillas se hará mediante las siguientes instrucciones donde entre comillas hacen referencia al nombre de la sección definida en app.edge.

Es decir, supongamos que se ha creado una nueva plantilla ejemplo.edge esta plantilla estará compuesta por el siguiente código donde @section('title') hace referencia a la sección definida en el fichero app.edge:

```
@layout('layouts.app')
@section('title')
   //código
@endsection
```

Implementación de las opciones multi idioma

Uno de los requisitos mencionados anteriormente es que la herramienta debe de estar en varios idiomas. Para ello, se utiliza el módulo i18n-express que nos permitirá manejar la opción de multi idiomas.

A continuación se muestra cómo se debe configurar el módulo, dicha configuración debe realizarse en el fichero index.js:

```
const i18n = require("i18n-express");

app.use(i18n({
      translationsPath: path.join(__dirname, 'i18n'),
      siteLangs: ["en","es"],
      textsVarName: 'translation',
      browserEnable: false,
      defaultLang: "en"
}));
```

- translationPath: Especifica la ruta de los archivos de traducción que se encontraran en la carpeta i18n. Se ha de tener en cuenta que la ruta se sitúa en el mismo nivel que el fichero index.js
- siteLang: Idiomas admitidos en la herramienta.
- textsVarName: Nombre de la variable que contiene las traducciones cargadas.
- browserEnable: Si está habilitado, intente obtener el idioma del usuario de los encabezados del navegador.
- defaultLang: idioma por defecto en caso de que los métodos anteriores fallen.

Para añadir un nuevo idioma, en la carpeta i18n, carpeta encargada de contener los archivos de traducción, se creará un nuevo fichero json que contendrá la traducción de todas las etiquetas creadas en los demás ficheros.

A continuación se muestra un ejemplo del contenido donde la sección roja indica el nombre de la etiqueta mientras la sección verde indica la traducción. Dicha etiqueta será la que se utilizará en el resto de ficheros para hacer referencia al texto que se desea mostrar.

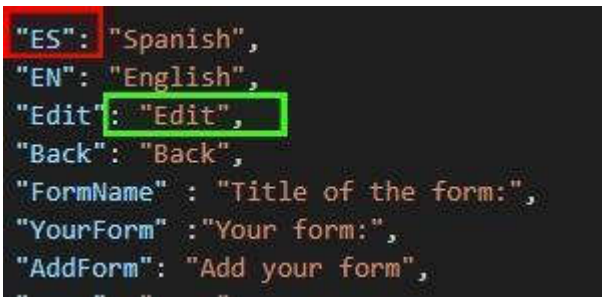


Ilustración 17 - Ejemplo fichero de idiomas

Finalmente, hay dos maneras de utilizar el sistema de multi idioma, de manera que en la interfaz final del usuario se muestre los textos correspondientes al idioma escogido por el usuario.

La primera manera implica la modificación de las plantillas edge, en las cuales se deberá de seguir la estructura de `{{translation.AddForm}}` donde translation es la variable definida en la configuración y AddForm la etiqueta que representa el texto que se desea mostrar.

La otra opción sucede cuando el texto varía dependiendo del contexto esto sucede en el caso de listar las reglas.

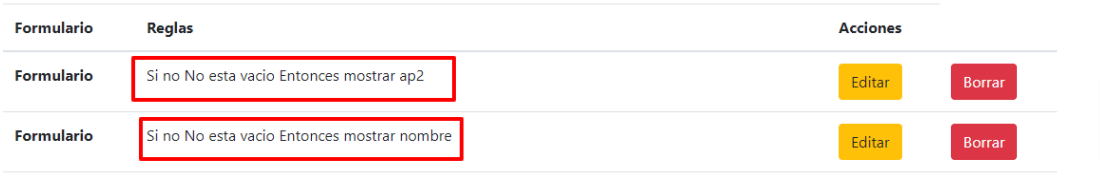


Ilustración 18 - Ejemplo texto multi idioma

Para ello en el fichero index.js, por cada regla que haya creada en el formulario se le asignará un texto, texto que será multi idioma, esta asignación se hará mediante la función generateRuleText().

```
rules.forEach(rule => {
        var conditionText = generateRuleText(rule,req);
        rule["conditionText"] = conditionText;
});
```

La función generateRuleText(rule, req) donde los parámetros rule y req representan la regla a la que se le asignará un texto y la petición al controlador respectivamente.

Por lo tanto, para generar la regla Si condicionante *No está vacío* entonces *mostrar/no mostrar/habilitar/obligatorio/mostrar bloque* condicional se generará de la siguiente manera:

```
if(rules.condition === 'nonempty'){
        conditionText = req.i18n_texts.If +" " + rules.inputAIdentifier + " " +
req.i18n_texts.NonEmpty +" ";
        if(rules.conditionThen === 'thenshow'){
                conditionText += req.i18n_texts.Then +" " +req.i18n_texts.ShowRule + " " +
rules.inputBIdentifier;
        }else if(rules.conditionThen === 'thenNotShow'){
                conditionText += req.i18n_texts.Then +" " +req.i18n_texts.NotShowRule + " "
+ rules.inputBIdentifier;
        }else if(rules.conditionThen === 'thenable'){
                conditionText += req.i18n_texts.Then +" " +req.i18n_texts.AbleRule + " " +
rules.inputBIdentifier;
        }else if(rules.conditionThen === 'thenunable'){
                conditionText += req.i18n_texts.Then +" " +req.i18n_texts.MandatoryRule + "
" + rules.inputBIdentifier;
        }else if(rules.conditionThen === 'showLock'){
                conditionText +=  req.i18n_texts.Then +" " +req.i18n_texts.ShowLock + " "  +
rules.showLockIdentifier;
        }
}
```

Por lo tanto, se puede observar que que para utilizar las variables de multi idioma sigue la siguiente estructura `req.i18n_texts.Then` donde req representa la petición

al controlador, i18n_text variable definida en el controlador donde almacena los ficheros lenguaje y finalmente Then etiqueta creada en los ficheros lenguaje.

Validación de código

La herramienta desarrollada en este TFE está diseñada para trabajar sobre código HTML válido, que cumpla la semántica y estructura oficial contemplada en el estándar HTML5.2. Por ello, se ha implementado una funcionalidad adicional de manera que a la hora de crear un formulario sea posible validar el código HTML introducido. Para ello, se utilizará el módulo html-validator el cual dispone de una función, *validator,* encargada de realizar la validación del código. El módulo se importara de la siguiente manera:

```
const validator = require('html-validator')
```

Por otro lado, para realizar la comprobación se necesitará definir un objeto de opciones el cual se definirá de la siguiente manera:

```
var options = {
  format: 'text',
  data: html,
  ignore: 'The document validates according to the specified schema(s).'
}
```

- Format representa el formato de los datos devueltos en este caso será tipo texto.
- Data, contendrá la URL o los datos que se quieren validar.
- Ignore: contendrá la respuesta que desea que el verificador elimine.

Inicialmente se hará una validación con las opciones descritas anteriormente mediante la instrucción siguiente:

```
var result = await validator(options);
```

Por un lado, si el código HTML no dispone de la instrucción “<!DOCTYPE html>” se mostrara un error indicando *"Error: Start tag seen without seeing a doctype first. Expected “<!DOCTYPE html>””*. No se quiere tener en cuenta este error por lo que habrá que crear un nuevo objeto de opciones especificando que se quiere ignorar

dicho error por lo que, después se volvería a validar para disponer de la validación final.

Finalmente, el código para validar un código HTML sería el siguiente:

```
var options = {
  format: 'text',
  data: html,
  ignore: 'The document validates according to the specified schema(s).'
}
var result = await validator(options);
if(result.indexOf("Error: Start tag seen without seeing a doctype first. Expected
“<!DOCTYPE html>”")==0){

  var options = {
    format: 'text',
    data:'<!DOCTYPE html>'+ html,
    ignore: 'The document validates according to the specified schema(s).'
  }
  var result = await validator(options);
}
if(result.indexOf("The document validates according to the specified
schema(s).")==0){
//validacion correcta
}
```

Obtención de input y sus atributos

En este apartado se describe la captura de la meta información de los inputs, es decir, atributos name, type, class, value se realiza mediante jquery esto se realizara en el fichero Procesor.js situado en la carpeta utils.

JQuery es una biblioteca de funciones y utilidades escritas en JavaScript que permite simplificar el modo de interactuar con las web así como de presentar la información en la misma.

Por lo tanto, para capturar dicha información se ha de hacer en varios pasos.

Primero, se importaran las librerías necesarias para utilizar JQuery:

```
const { JSDOM } = require( "jsdom" );
const { window } = new JSDOM( "" );
const $ = require( "jquery" )( window );
```

Después, se ha de parsear el formulario mediante la instrucción *$.parseHTML(textform)* donde textform es el formulario del que se quiere obtener la información.

ParseHTML es un método para convertir la cadena en un conjunto de nodos DOM, que luego se pueden insertar en el documento. Los nodos Dom son una representación interna del código HTML como un árbol de nodos y a esto lo denominamos DOM (Document Object Model). Los nodos se pueden clasificar en diferentes tipos, dentro de los cuales los principales tipos son: document, element, text y "otros". Este método se usará para recorrer los nodos DOM del formulario y obtener los elementos de dicho formulario.

Un vez parseado el formulario se añadirá un nodo DOM, al código HTML para poder encontrar los elementos <form> que haya en el formulario de esa manera se podrá obtener los input del formulario seleccionado, ya que una de las opciones que dispone la herramienta es seleccionar un formulario de todos los que haya en el código HTML.

```
var formhtml = $.parseHTML(textform);

var container = $.parseHTML("<div></div>");
$(container).append(formhtml);
```

Para recorrer el código HTML parseado se hará mediante la instrucción:

```
$(container).find("form").each(function(formNum, formElem){
  $(formElem).find("input,select,textarea").each(function(inputNum,inputElem){
  }
}
```

La cual indica que se ha de encontrar las etiquetas form y por cada etiqueta form encontrada buscar los elementos input, select y textarea.

Después se obtendrá los atributos name, type, class y value para cada elemento encontrado mediante las instrucciones:

```
var ids = $(this).attr("id");
var names = $(this).attr("name");
var classes = $(this).attr("class");
```

```
var values = $(this).attr("value");
```

Por otro lado, los input que sean de tipo hidden o submit no se han de tener en cuenta para ello, mediante las siguientes instrucciones prescindiremos de dichos input:

```
if(types === "hidden"){
  return true;
}else if(types === "submit"){
  return true;
}
```

Posteriormente, se obtendrá el querySelector del input analizado. Como se ha mencionado anteriormente mediante querySelector se accedera al elemento por su colección etiquetas utilizando un recorrido primero en profundidad pre ordenado de los nodos del documento.

```
var elementPosition = $(this).index();elementPosition = elementPosition +1;

var elementAux = $(this).parent();
var elementType = tag;
var querySelector =  elementType+ ":nth-child("+elementPosition+") ";
var parentPosition = elementAux.index();
var parentType = elementAux.prop('nodeName');

while( elementAux != undefined && parentType !== "FORM" && parentType !=
undefined && parentType !== "undefined" && parentPosition>=0){

   parentPosition = elementAux.index();
   parentPosition = parentPosition + 1 ;
   parentType = elementAux.prop('nodeName');

   if (parentType === 'FORM'){
      if (parentPosition != 1){
         parentPosition = parentPosition -2;
      }
   }
   querySelector = parentType.toLowerCase() +":nth-child("+ parentPosition +")" +" >
"+ querySelector;

   elementAux = elementAux.parent();
}
```

A continuación, se asigna un identificador a cada input ya bien sea por el id, name o class de dicho input y en caso de que no disponga ninguno de ellos se le asignará un nombre mediante la función getNameForInput().

```
if((ids === undefined) && (names === undefined) && (classes === undefined) ){
    var indexOfarrayTag = arrayTag.indexOf(tag);
    inputIdentifiers = getNameForInput(inputoptionS,indexOfarrayTag);
}else if(ids !== undefined){
    inputIdentifiers = ids;
}else if(names !== undefined){
    inputIdentifiers = names;
}else if(classes !== undefined){
    inputIdentifiers = classes;
}
```

El nombre que se asigna será el tag del input a tratar junto con el número de elementos que hay con el mismo tag.

Es decir, supongamos que tenemos el siguiente ejemplo:

```
<html>
<head>
</head>
<body>
<form id="formulario1">
        <div>
                <input type="text" >
        </div>
</form>
</body>
```

El input no tiene ningún id, name o class por lo tanto, mediante la función getNameForInput() se le asignará un identificador para poder diferenciarlo a la hora de crear reglas, en este caso, el identificador asignado será INPUT_0 ya que únicamente hay un INPUT.

Finalmente, se guardará el input con todos los datos recogidos en la base de datos.

Generar javaScript con las reglas almacenadas

En este apartado se define cómo generar el javaScript con las reglas almacenadas para un formulario y se realizará en el fichero Events.js en la carpeta utils.

Inicialmente se necesitará un proceso de carga del documento es decir, cuando todos los objetos del DOM (imágenes, flash, scripts, frames) hayan terminado de cargarse se aplicará el javaScript generado. Esto se realiza mediante la siguiente instrucción:

```
window.onload = function() { addSmartFormEvents(); };
```

Donde addSmartFormEvents() contendrá la llamada a las funciones generadas por cada regla.

Por otro lado, se ha de mencionar que cada regla estará compuesta por dos funciones, initialStatus_Rule0Event(), la cual contendrá el estado inicial de los input, y la función onChange_Rule0Event(), la cual se encargará de realizar los cambios necesarios en los inputs.

Por lo tanto, para definir la base del javaScript se realizará de la siguiente manera generando en la variable codeBase el resultado del javaScript correspondiente:

```
var codeBase = "window.onload = function() { addSmartFormEvents(); };\n";
codeBase = codeBase + "function addSmartFormEvents(){";

for(var i = 0; i<arrayInputRule.length; i++){

   var inputName = inputsName[i];

   var functionText =
"document.querySelector('"+inputName+"').addEventListener('change',
onChange_Rule_"+arrayInputRule[i]+"Event);";
   functionText += "initialStatus_Rule_"+arrayInputRule[i]+"Event();";
   codeBase += functionText;
}
codeBase += "}\n";
```

Una vez se disponga de la base, se crearán las funciones initialStatus_Rule0Event() y onChange_Rule0Event() mencionadas para cada regla.

Primero, habrá que distinguir si el condicionante de la regla a tratar es de tipo checkbox, radio u otro ya que cada uno de ellos se tratara de distinta manera. Por ello, mediante la funcion getInputType(identificador_formulario, condicionante_regla); se obtendrá el type del input condicionante.

Supongamos que se tiene la regla Si condicionante *No está vacío* entonces *mostrar* condicional, donde el condicionante es de tipo radio.

Para generar el javaScript de esta regla se haría mediante el siguiente código:

```
if(rule.condition === 'nonempty'){ //if inputA is not empty
        if(rule.conditionThen === 'thenshow'){//then show inputB
            jsCode += "var label = document.querySelector('label[for=\""+rule.inputBIdentifier +
"\"]');";
            jsCode += "if(label!=null){";
            jsCode += "if(document.querySelector('"+rule.inputA+"').checked){";
            jsCode += "document.querySelector('label[for=\""+rule.inputBIdentifier +
"\"]').style.display = 'block';";
            jsCode += "document.querySelector('"+rule.inputB+"').style.display = 'block';";
            jsCode += "}  ";

            if( querySelectorAllRadio.length>0){
                for(var i=0;i < querySelectorAllRadio.length;i++ ){
                    jsCode +=
"if(document.querySelector('"+querySelectorAllRadio[i]+"').checked){";
                    jsCode += "document.querySelector('label[for=\""+rule.inputBIdentifier +
"\"]').style.display = 'none';";
                    jsCode += "document.querySelector('"+rule.inputB+"').style.display = 'none';";
                    jsCode += "}";
                }
            }

            jsCode += "}else{";

            jsCode += "if(document.querySelector('"+rule.inputA+"').checked){";
            jsCode += "document.querySelector('"+rule.inputB+"').style.display = 'block';"
            jsCode += "}  ";

            if( querySelectorAllRadio.length>0){
                for(var i=0;i < querySelectorAllRadio.length;i++ ){
                    jsCode +=
"if(document.querySelector('"+querySelectorAllRadio[i]+"').checked){";
                    jsCode += "document.querySelector('"+rule.inputB+"').style.display = 'none';";
                    jsCode += "}";
                }
            }

            jsCode += "}";

            initialStatus += " document.querySelector('"+rule.inputB+"').style.display = 'none';";
            initialStatus += "var label =
document.querySelector('label[for=\""+rule.inputBIdentifier + "\"]');";
            initialStatus += "if(label!=null){";
            initialStatus += "document.querySelector('label[for=\""+rule.inputBIdentifier +
"\"]').style.display = 'none';";
        initialStatus += "}"
        }
}
```

En este código, rule.condition representa la acción que debe cumplir el condicionante, en el ejemplo puesto sería No está vacío, y rule.conditionThen la acción a cumplir por el condicional, en el ejemplo sería mostrar.

Buscador de formularios

Una de las opciones disponibles es buscar el formulario deseado como se muestra en la imagen y además muestra el número de coincidencias encontradas. La implementación del buscador se ha realizado en el fichero Function.js

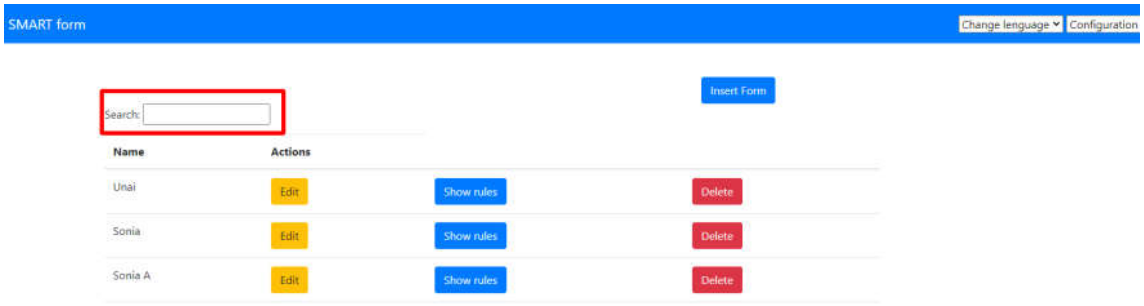


Ilustración 19 - Buscador de formularios

Para implementarlo se ha realizado mediante el siguiente código donde por cada modificación en el campo de búsqueda se transforma el texto que contiene a minúscula para compararlo con todas las filas de la tabla, de manera que no haya problemas entre las mayúsculas y minúsculas, después se retiran las filas que no hayan coincidido con el campo de búsqueda y por el contrario, se añade las filas que coincidan aplicando los estilos necesarios.

```
function doSearch(x, SearchMatch, SearchNoMatch)
{
    const tableReg = document.getElementById('example');
    const searchText =
document.getElementById('searchTerm').value.toLowerCase();
    let total = 0;

    for (let i = 1; i < tableReg.rows.length; i++) {
        if (tableReg.rows[i].classList.contains("noSearch")) {
            continue;
        }
        let found = false;
        const cellsOfRow = tableReg.rows[i].getElementsByTagName('td');
        for (let j = 0; j < cellsOfRow.length && !found; j++) {
            const compareWith = cellsOfRow[j].innerHTML.toLowerCase();
            if (searchText.length == 0 || compareWith.indexOf(searchText) > -1) {
                found = true;
                total++;
            }
        }
        if (found) {
            tableReg.rows[i].style.display = '';
        } else {
            tableReg.rows[i].style.display = 'none';
```

```
        }
    }

    const lastTR=tableReg.rows[tableReg.rows.length-1];
    const td=lastTR.querySelector("td");
    lastTR.classList.remove("hide", "red");
    if (searchText == "") {
        lastTR.classList.add("hide");
    } else if (total) {
        td.innerHTML= total + SearchMatch;
    } else {
        lastTR.classList.add("red");
        td.innerHTML= SearchNoMatch;
    }
}
```

Buscador select2

A la hora de definir reglas se ha implementado un buscador de inputs. Este buscador se ha realizado mediante la opción Select2 plugin que extiende las funcionalidades de jquery.

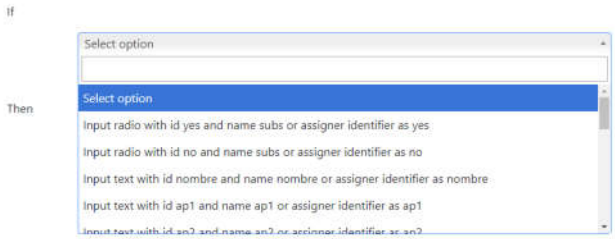


Ilustración 20 - Buscador select2

La configuración del buscador select2 se realiza en el fichero Functions.js:

```
function searchSelect2(){

    $("#inputS").select2({
        theme: "classic"
    });
    $("#inputS2").select2({
        theme: "classic"
    }).next().hide();
    $("#showLockResult").select2({
        theme: "classic"
    }).next().hide();
    $("#thenoption").select2({
        theme: "classic"
    }).next().hide();
```

```
}
```

En total se han realizado cuatro buscadores select2 por lo tanto, las etiquetas #inputS,#inputS2, #showLockResult y #thenoption representan los identificadores de los input donde se utiliza el buscador.

Por otro lado, para ocultar el select2 debe hacerse mediante .next().hide() inicialmente deben estar ocultos ya que se mostrarán dependiendo de cómo se definan las reglas, y para mostrarlo se realizará mediante la sentencia .next().show() especificando la clase del input en el cual está el buscador que se desea mostrar

```
jQuery(".inputB").select2().next().show();
```

Problemáticas detectadas

Uno de los mayores problemas encontrados a la hora de realizar la herramienta es el conflicto de reglas. La herramienta provee a la persona usuaria de suficiente libertad para modelar y acumular múltiples reglas que pueden llegar a resultar conflictivas entre sí. Una persona usuaria despistada podría diseñar un sistema de reglas paradójico que propusiera aplicar reglas o efectos contradictorios, encontrándose así el usuario con comportamientos inesperados o percibidos como erróneos al no contemplar el sistema de reglas en su completitud.

Ejemplificamos suponiendo que tenemos las siguiente reglas creadas:

- Regla 1: Si input nombre *No esta vacío* entonces *mostrar* input Apellido1
- Regla 2: Si input nombre *Está vacío* entonces *mostrar* input Apellido1

Sucederían los siguientes casos:

Caso 1: cuando el input nombre está vacío se muestra Apellido1

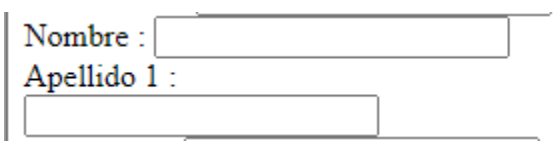


Ilustración 21 - Ejemplo problema encontrado caso 1

Caso 2: cuando input nombre no está vacío no se muestra Apellido 1

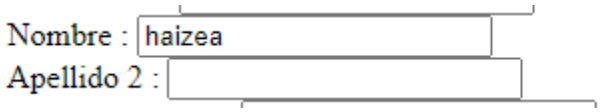


Ilustración 22 - Ejemplo problema encontrado caso 2

Por lo tanto, se puede comprobar que hay un conflicto de reglas ya que no se puede dar el caso de mostrar Apellido 1 cuando el input Nombre esté o no vacío por lo que, únicamente se aplica la Regla 2.

Esto sucede por la forma en la que se ha codificado el JavaScript, es decir, a la hora de generar el JavaScript inicialmente se invocan dos funciones para inicializar el estado de los input, funciones initialStatusRule_0Event() y initialStatusRule_1Event(). Cada una de estas funciones manejadoras corresponde a cada una de las reglas modeladas que afectan al elemento sobre las que se aplican y a su cambio de estado. Javascript permite asignar dos o más funciones manejadoras para cierto evento mediante la función addEventListener() y estas se ejecutarán secuencialmente. Nuestro problema surge cuando estos manejadores afectan a la misma propiedad, ya que únicamente persistirá la última modificación realizada.

```
window.onload = function() {
    addSmartFormEvents();
};

function addSmartFormEvents() {
    document.querySelector('form:nth-child(1) > div:nth-child(4) > input:nth-child(2) ').addEventListener('change', onChange_Rule_0
Event);
    initialStatus_Rule_0Event();
    document.querySelector('form:nth-child(1) > div:nth-child(4) > input:nth-child(2) ').addEventListener('change', onChange_Rule_1
Event);
    initialStatus_Rule_1Event();
}
```

Ilustración 23 - Ejemplo problema encontrado conflicto de reglas

En nuestro ejemplo, estas funciones modifican el estado del input Apellido1, escondiendo y mostrando el input respectivamente por lo que, finalmente el input Apellido1 se estaría mostrando ya que initialStatusRule_1Event() estaría sobrescribiendo a la regla anterior.

```
function initialStatus_Rule_0Event() {
    document.querySelector('form:nth-child(1) > div:nth-child(5) > input:nth-child(2) ').style.display = 'none';
    var label = document.querySelector('label[for="ap1"]');
    if (label != null) {
        document.querySelector('label[for="ap1"]').style.display = 'none';
    }
}

function initialStatus_Rule_1Event() {
    document.querySelector('form:nth-child(1) > div:nth-child(5) > input:nth-child(2) ').style.display = 'block';
    var label = document.querySelector('label[for="ap1"]');
    if (label != null) {
        document.querySelector('label[for="ap1"]').style.display = 'block';
    }
}
```

Ilustración 24 - Ejemplo problema encontrado sobrescribir regla

Por lo tanto, cuando sucede un conflicto de reglas únicamente se aplicará la última regla aplicada sobre las reglas que generan conflictos.

Por otro lado, el select2 utilizado para crear un buscador de input a la hora de definir reglas no funcionaba siempre, es decir, a la hora de seleccionar un input no se seleccionaba correctamente por lo que era costoso o incluso creaba mal las reglas.

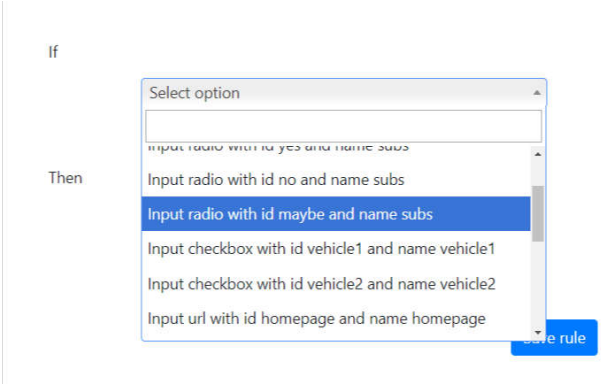


Ilustración 25 - Ejemplo problema select2

Inicialmente el buscador del selector no se cargaba correctamente nada más seleccionar un selector había que seleccionarlo varias veces. Para solucionarlo se creó una función que cargaba los buscadores nada más cargar la página, es decir, en el <body> fue necesario especificar onload, elemento utilizado para ejecutar un script una vez que la página web ha cargado completamente todo el contenido. Teniendo que jugar con las funciones de hide() y show() para mostrar el buscador y de esa manera se evitaba el tener que seleccionar varias veces el input deseado.

```
function myFunction(){

    $("#inputS").select2({
        theme: "classic"
    });
    $("#inputS2").select2({
        theme: "classic"
    }).next().hide();
    $("#showLockResult").select2({
        theme: "classic"
    }).next().hide();
    $("#thenoption").select2({
        theme: "classic"
    }).next().hide();

}
```

Ilustración 26 - Ejemplo solución al problema select2

Finalmente, a la hora de realizar prueba de usuario se encontró que el orden para comprobar si las reglas estaban correctamente importaba. Es decir, supongamos que se tiene la regla Si las contraseñas *No coinciden* entonces *mostrar* input Mensaje

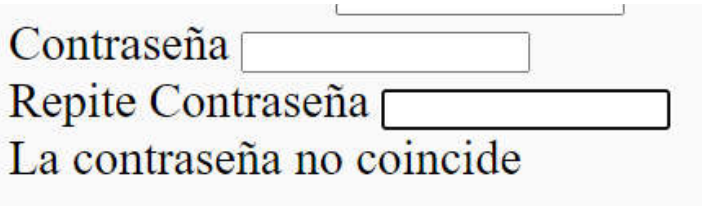


Ilustración 27 - Ejemplo orden de reglas

Para que la regla funcionara correctamente el orden a introducir sería primero Repite Contraseña y después, contraseña, de lo contrario la comparación no se realizaba correctamente.

En esta sección se pretende mostrar el funcionamiento correcto de la herramienta mediante capturas, así mismo, se ha de mencionar que en la sección Anexo A - Manual de usuario se dispone el funcionamiento de toda la herramienta en completitud mostrando detalles concretos de la herramienta resultante del proyecto.

Inicialmente, se tendrá una página de logeo para poder acceder a la herramienta.

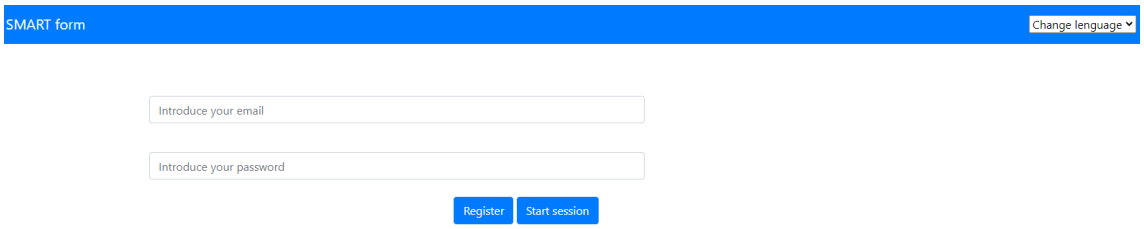


Ilustración 28 – Inicio de sesión

Una vez iniciado sesión en la herramienta se podrá comenzar con la creación/edición/eliminación de formularios.

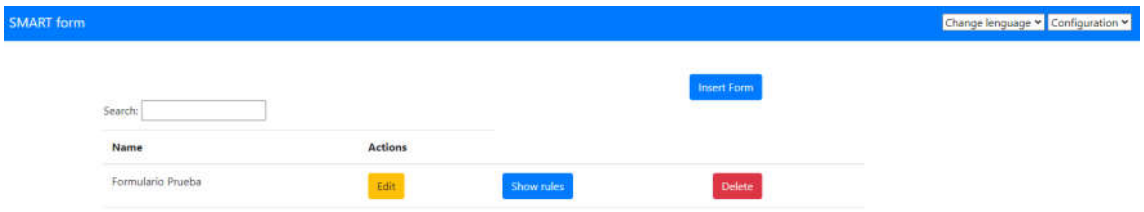


Ilustración 29 Listado de formularios

Respecto a la creación de un formulario se podrá obtener el código mediante dos formas, la primera, mediante una URL, la segunda, insertando el código manualmente. Por otro lado, se dispondrá de la opción de validar el código HTML el cual por defecto estará activado.

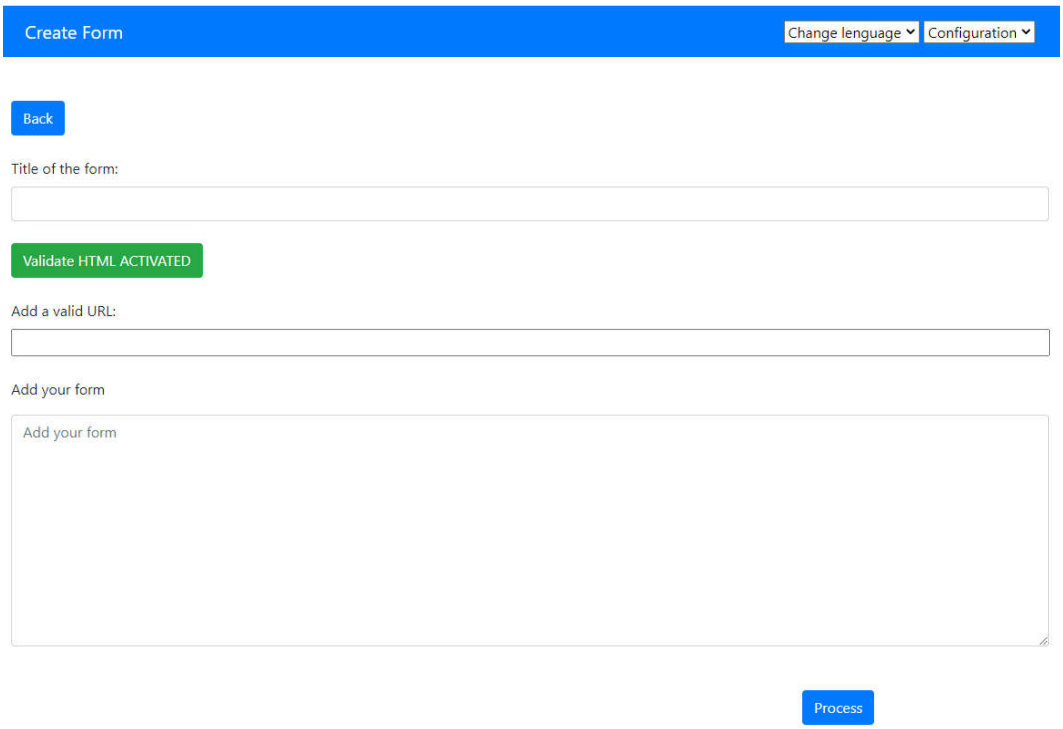


Ilustración 30 – Creación de formularios

Si en el código HTML se encuentran más de un formulario se dispondrá de la opción de seleccionar sobre qué formulario se desea trabajar, dando una pre-visualización del formulario.

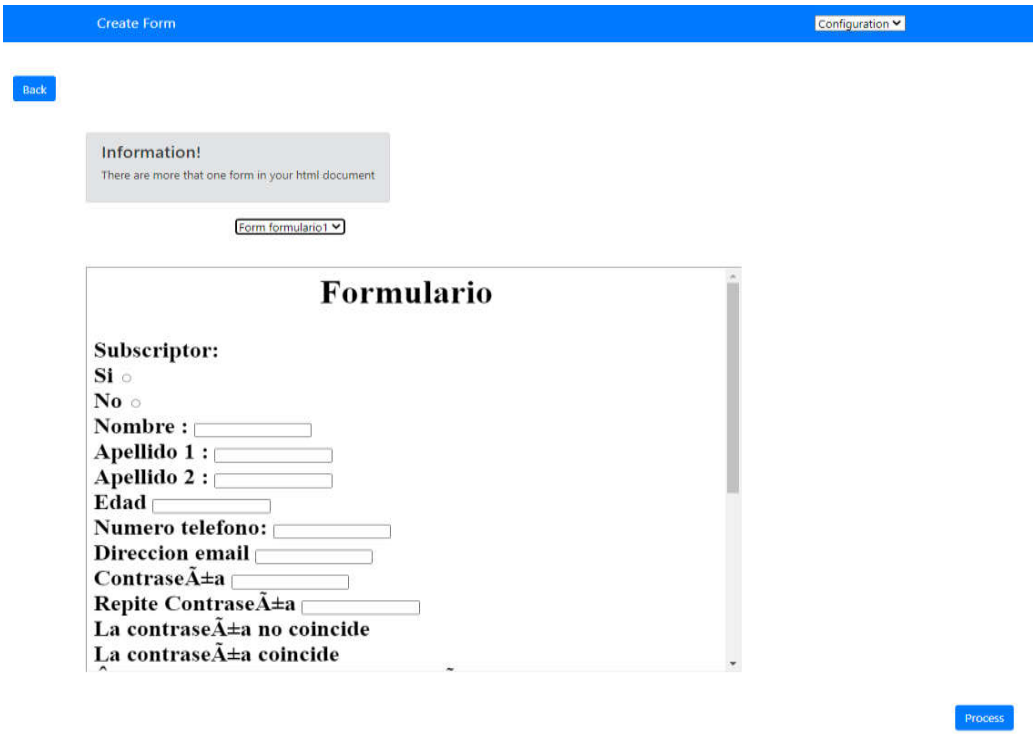


Ilustración 31 – Selección de formulario

Respecto a la edición del formulario, se hará de la misma forma que la creación, teniendo la opción de validación de código, la única diferencia es que únicamente se podrá editar el formulario si no hay reglas creadas con anterioridad, de lo contrario se indicará que hay reglas creadas y que no se puede realizar la edición del formulario.

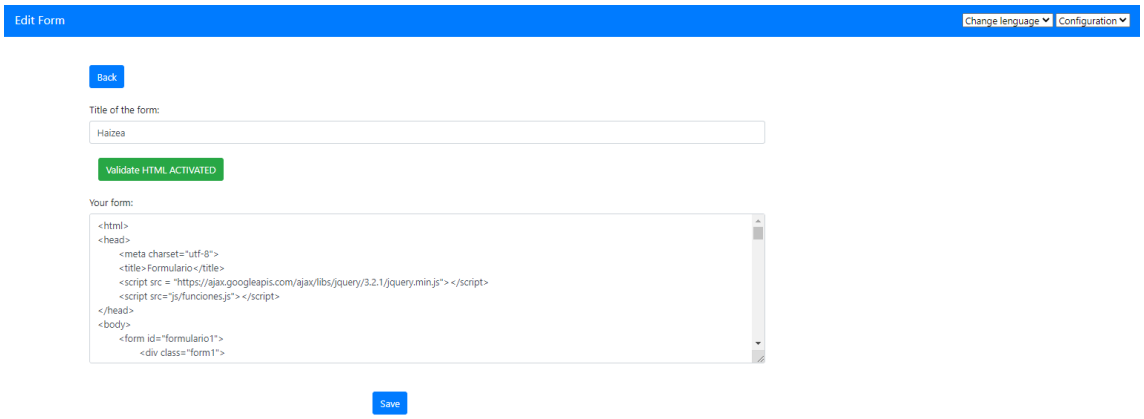


Ilustración 31 – Editar formulario

Respecto a las reglas, se tendrá un listado de las reglas creadas hasta el momento y la opción de generar código, la cual se comentará más adelante.

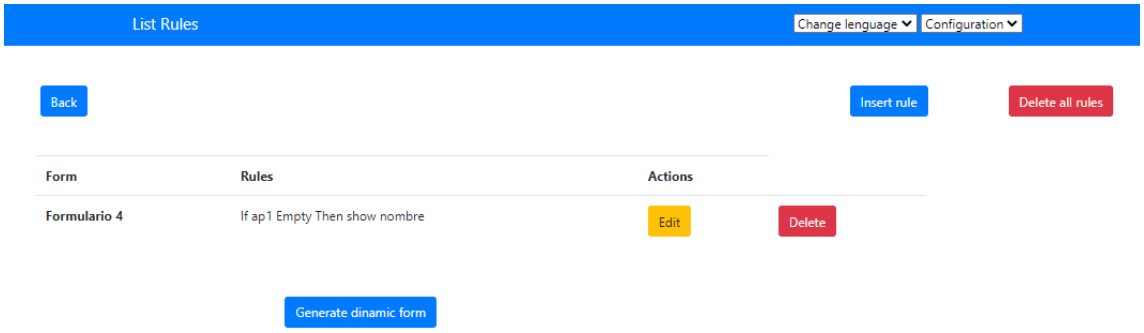


Ilustración 32 – Listado de reglas

Respecto la creación / edición de reglas, la estructura que sigue una regla es la siguiente: Si condicionante entonces determinante. Un ejemplo sencillo sería: Si el input con el id nombre no está vacío entonces mostramos el input con el id ap1.

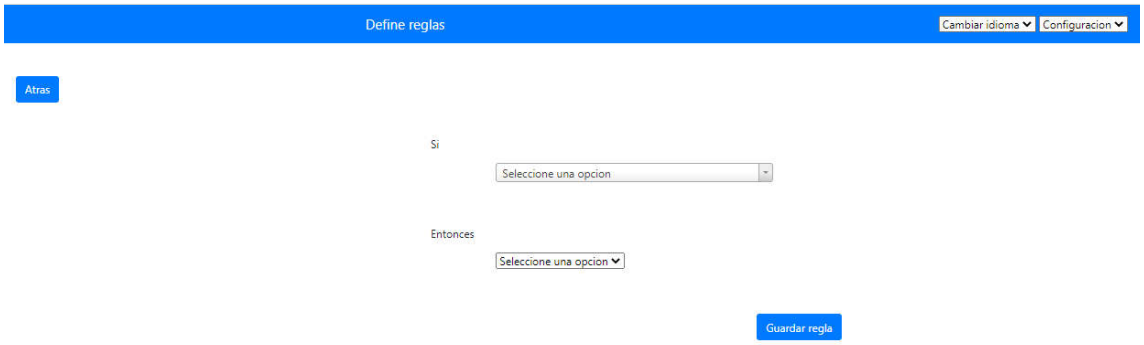


Ilustración 33 – Creación de reglas

Teniendo como condicionantes disponibles Es igual, No es igual a, Contiene, Esta vacio o esta seleccionado, No está vacío o no está seleccionado y Entre.

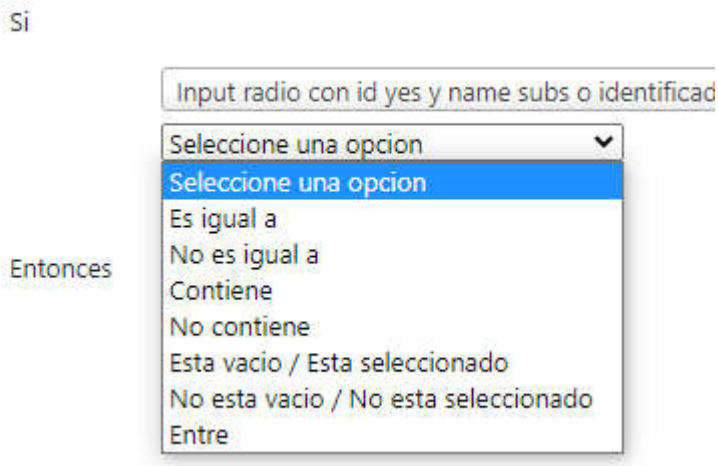


Ilustración 34 – Condicionantes

Y respecto al determinante las opciones disponibles son Mostrar, No mostrar, Habilitar un campo, Hacer un campo obligatorio o Mostrar bloque. Se ha de mencionar que la opción Mostrar bloque, sirve para mostrar un conjunto de campos a la vez

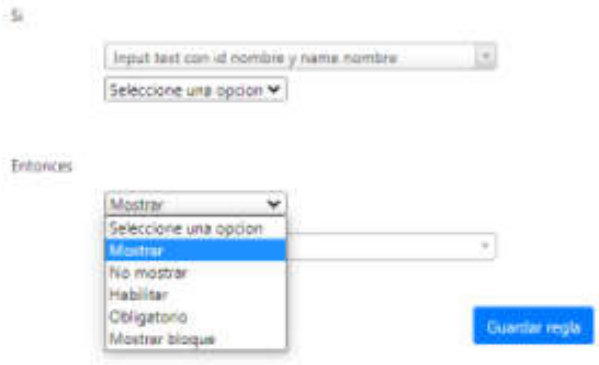


Ilustración 34 – Determinantes

Respecto a la eliminación de reglas se dispondrá de la opción de eliminar todas las reglas a su vez o de eliminar una regla en concreto como se muestra en la siguiente captura.

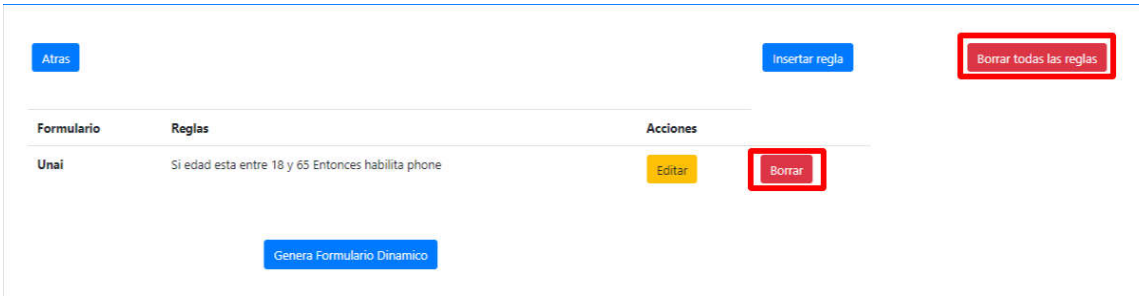


Ilustración 35 – Eliminación de reglas

Por otro lado, una vez generadas las reglas se dispone de la opción de Generar formulario dinámico.

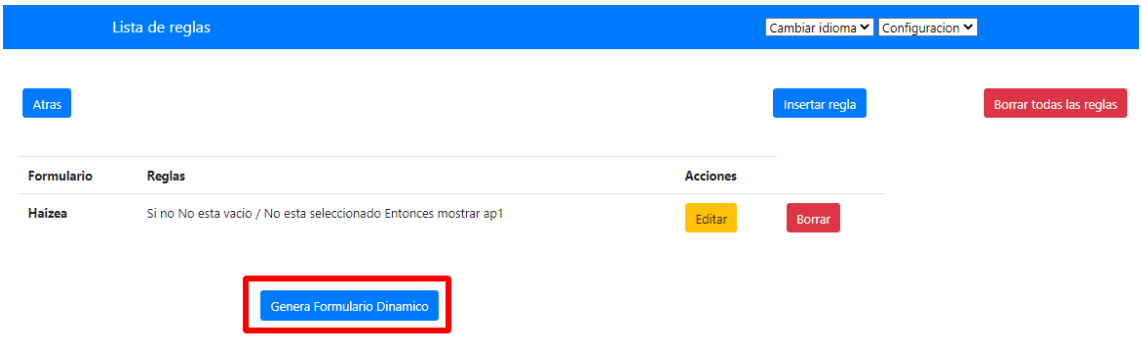


Ilustración 36 – Generación de formulario dinámico

Esta opción generará el código JavaScript con las reglas creadas. Además, se podrá copiar el código manualmente o exportarlo como documento JS o HTML.

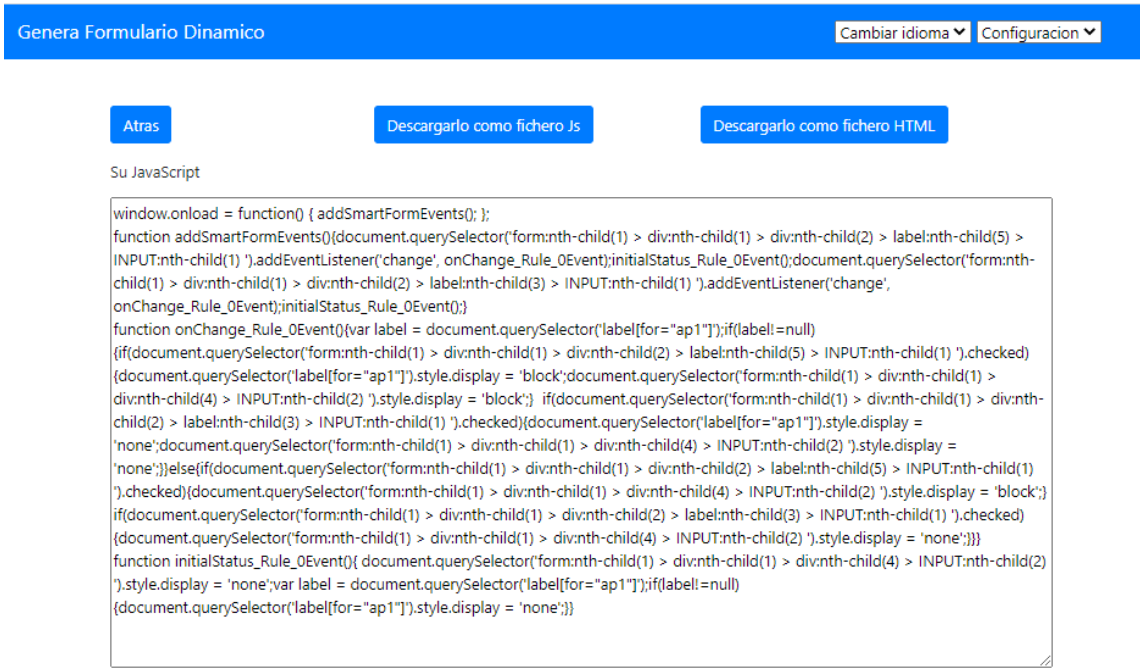


Ilustración 37 – Formulario dinámico

Finalmente, el usuario dispondrá de las opciones de configurar su perfil o cambiar el idioma de la herramienta siempre que desee. Los idiomas disponibles actualmente son Español e Inglés.

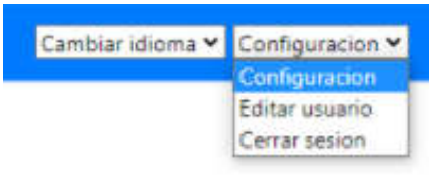


Ilustración 38 – Configuración de un usuario

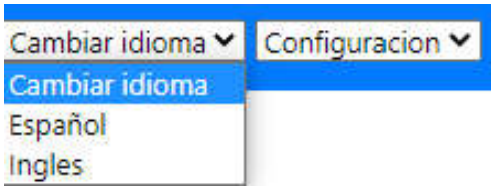


Ilustración 39 – Cambiar idioma

Además de que la herramienta exista se ha comprobado que funciona y aporta ventajas realizando pruebas a usuarios, las pruebas a usuarios se encuentra en la sección Pruebas de usuario.

Se pretende evaluar la eficacia, eficiencia y satisfacción con la que los usuarios logran concretar objetivos específicos utilizando la herramienta además de evaluar la usabilidad de la herramienta.

Para ello se van a tomar varias métricas y se realizará un cuestionario disponible en la sección Anexo B Pruebas de usuario además en dicho anexo también estará disponible el formulario que los usuarios han utilizado a la hora de realizar pruebas y los resultados individuales anonimizados.

Este cuestionario se compone de una serie de tareas a partir de las cuales tomar métricas objetivas como TCT (Task Completion Time) porcentaje de éxito o nº de consultas a documentación. El cuestionario se completa con una breve encuesta basada en el cuestionario de Nasa TLX. Nasa TLX es una herramienta de evaluación multidimensional que perciben tasas de carga de trabajo con el fin de evaluar la eficacia de una tarea, sistema o del equipo o de otros aspectos del rendimiento.

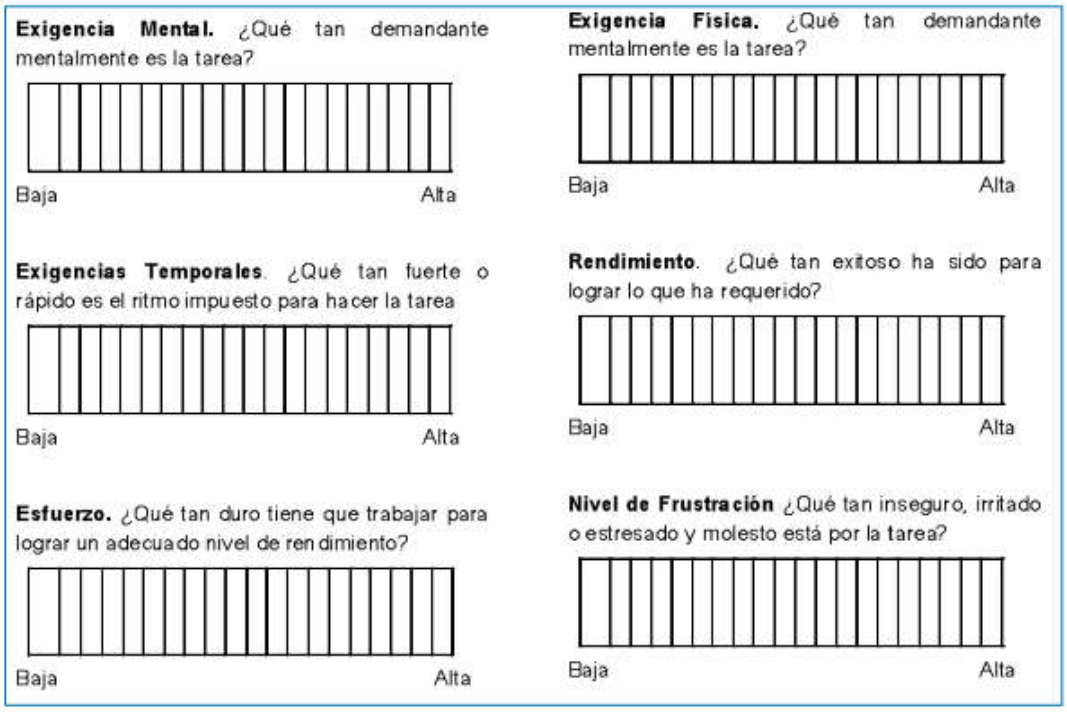


Ilustración 40 - Cuestionario Nasa TLX

Para evaluar la eficiencia de la herramienta se han planteado varios retos a cada usuario y además deberán realizar dicho cuestionario NASA TLX para cada reto:

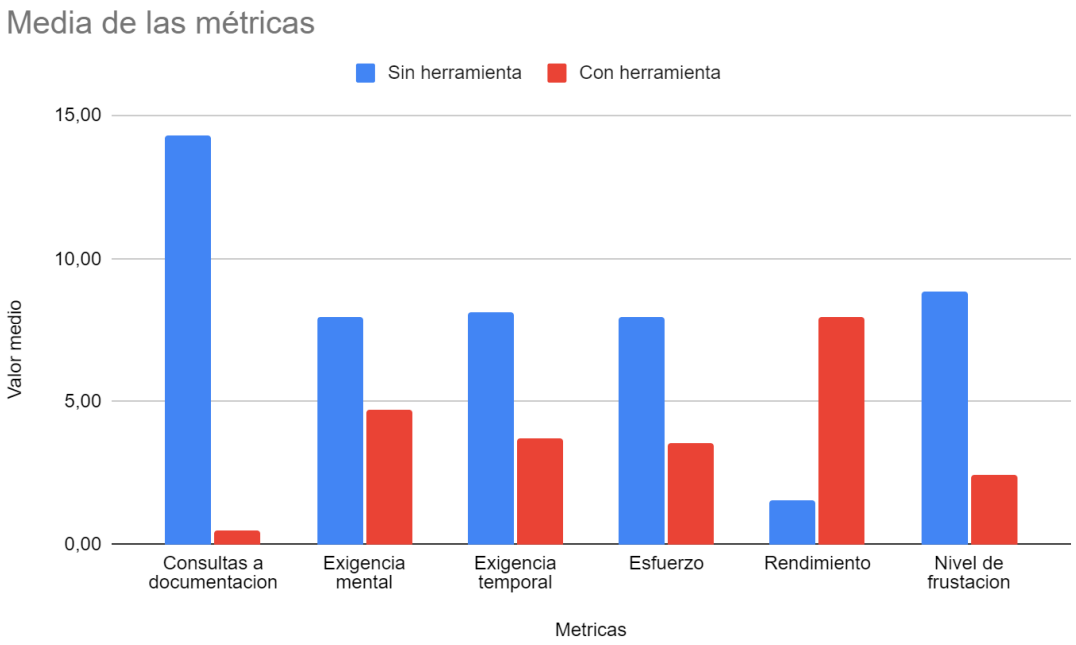
- Reto 1: dado un formulario HTML realizar unas reglas específicas utilizando mi herramienta.
- Reto 2: dado un formulario HTML realizar unas reglas específicas sin utilizar mi herramienta.

Las tareas que los usuarios han tenido que realizar y así como la documentación que han recibido esta especificada en la sección Anexo B Pruebas de usuario. Estas tareas se han escogido ya que son las más comunes al desarrollar formularios y además porque son las más problemáticas, entre ellas el manejo de los checkbox y radio.

A parte de extraer la eficiencia de la herramienta en base los éxitos también se extraerá el tiempo que tarda el usuario en realizar los retos, cantidad de enlaces de ayuda o consultas que ha necesitado para lograr el resultado.

A continuación se mostrarán las gráficas obtenidas después de analizar las métricas obtenidas. Se ha de mencionar que dichas pruebas se obtuvieron de 6 participantes los cuales la mitad tenían conocimiento avanzado sobre javaScript y la otra mitad no tenían ningún conocimiento previo sobre javaScript así mismo, se ha equilibrado el orden de los retos y participantes para evitar sesgos estadísticos. Es decir, la mitad de los participantes han comenzado con una herramienta y la otra mitad con otra, alternando para que entre ambos grupos hubiese equilibrio de participantes con y sin conocimientos de programación.

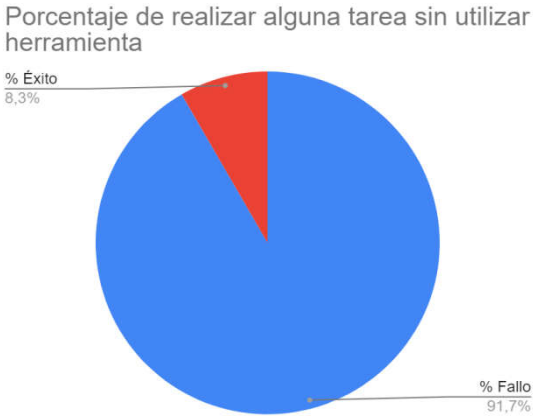
Por otro lado, se ha de remarcar que los niveles medidos por NASA TLX tienen un valor máximo de 10.



Gráfica 1 - Métricas

En esta gráfica, se puede llegar a la conclusión de que no se necesitan conocimientos previos para utilizar la herramienta y de que la herramienta resulta una alternativa significativamente más intuitiva a la programación tradicional, ya que el rendimiento de lograr los retos es mayor utilizando la herramienta. Además, la consulta de documentación externa es menor utilizando la herramienta y el esfuerzo requerido para realizar los retos es menor.

Por otro lado, mediante la métrica de rendimiento se obtiene que la satisfacción de lograr las tareas es mayor que sin utilizar la herramienta. Para analizar mejor esta métrica se ha realizado una gráfica con el porcentaje de acierto de realizar alguna de las tareas propuestas sin utilizar y utilizando la herramienta.

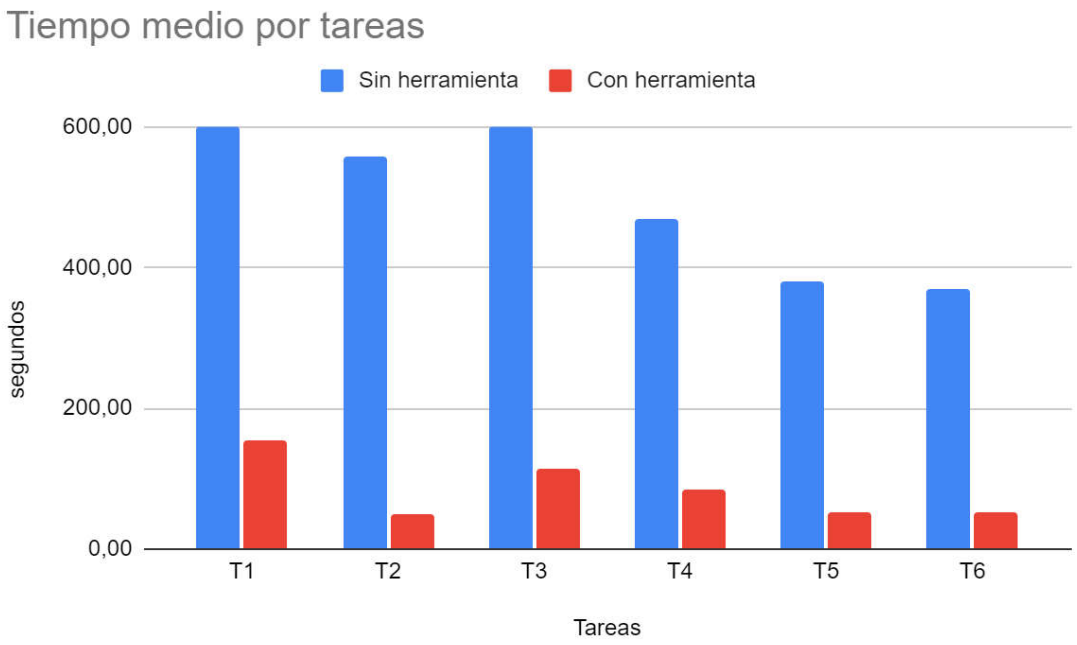


Gráfica 2 - Porcentaje de éxito al realizar tareas mediante codificación tradicional.

Gráfica 3 –Porcentaje de éxito al realizar tareas mediante la herramienta propuesta.

Por lo tanto, se puede deducir que la herramienta es eficaz a la hora de lograr las tareas propuestas ya que el porcentaje de realizar alguna tarea con éxito utilizando la herramienta es del 88.9% mientras que sin utilizar la herramienta es del 8.3%.

Finalmente, respecto a la exigencia temporal se puede observar en la gráfica 1 que el tiempo requerido para realizar los retos es menor utilizando la herramienta. Para una mejor visualización se ha realizado una gráfica con el tiempo medio requerido para cada una de las tareas, se ha de mencionar que el tiempo disponible para realizar cada tarea era de máximo diez minutos para agilizar el proceso de pruebas y para que a los participantes no les resultara tedioso por lo que si un participante se rindió al realizar una tarea se le asignó el máximo tiempo disponible, en este caso, diez minutos.



Gráfica 4 –Tiempo medio por tarea

Por lo tanto, mediante la gráfica anterior se confirma que mediante la herramienta el tiempo se reduce considerablemente cumpliendo el objetivo principal de la herramienta, facilitar la implementación de un código JavaScript reuniendo todas las reglas configuradas.

Finalmente, algunas de las observaciones que los participantes dieron son las siguientes:

- Dificultad en la comprensión de realizar reglas con inputs bloque ya que no queda muy claro el nombre asignado a que input hace referencia.
- Sencilla de utilizar, no requiere conocimientos previos para poder manejar la herramienta.
- La herramienta cumple con el objetivo de agilizar el proceso de creación de reglas en JavaScript ya que sin la herramienta el proceso es tedioso y de larga duración.

Conclusiones

En este capítulo se muestran las conclusiones obtenidas y las mejoras que se pueden realizar.

En primer lugar, el aprendizaje, así como la experiencia adquirida durante el desarrollo del proyecto ha sido considerablemente mayor de la que se esperaba inicialmente. A lo largo del proyecto se han adquirido nociones de nivel medio/avanzado sobre cómo implementar una herramienta utilizando JavaScript uno de los objetivos personales.

La herramienta desarrollada cumple con los requisitos funcionales estipulados al comienzo del proyecto, como son la capacidad de extraer metainformación sobre múltiples formularios, y diseñar reglas para múltiples tipos de input. Estas reglas pueden ser almacenadas y personalizadas mediante un sistema de usuarios y se permite su extracción mediante scripts Javascript. Además, se ofrece un sistema multilenguaje que puede ser ampliado con relativa facilidad.

Por último, las pruebas de usuario realizadas verifican la utilidad de la herramienta y su conveniencia de cara al uso interno por parte de la empresa. Por todo ello, considero que los objetivos técnicos han sido satisfechos con solvencia.

Así pues, se ha cumplido con los requisitos y objetivos propuestos inicialmente, consiguiendo una aplicación eficiente, plenamente funcional intuitiva, de alto rendimiento y que facilite el trabajo a aquellos quienes deban utilizar la herramienta asiduamente.

Trabajo futuro

Algunas de las mejoras que se propone realizar en el proyecto son:

- Tratar los conflictos de reglas expuestos en la sección dedicada a problemáticas.
- Poder escoger clicando sobre la visualización los inputs a los que aplicar las reglas. Es decir, en vez de tener un selector de inputs tener la opción de visualizar el formulario y sobre dicha visualización escoger los input para realizar las reglas.

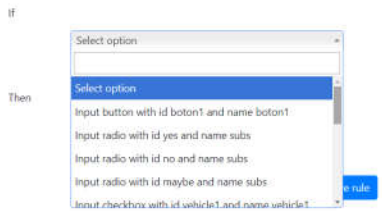


Ilustración 41 - Trabajo futuro visualización de input

Ilustración 42 - Trabajo futuro

- Definir los nombres del input bloque de otra manera. Es decir, actualmente los inputs bloque que no tengan los atributos name, id o class definidos se muestran como el tipo_Contador donde el contador asignado sería la posición del input. Esta manera de definir los nombres no es muy clara a la hora de realizar reglas ya que se tiene que estar mirando el código HTML y contando los input.

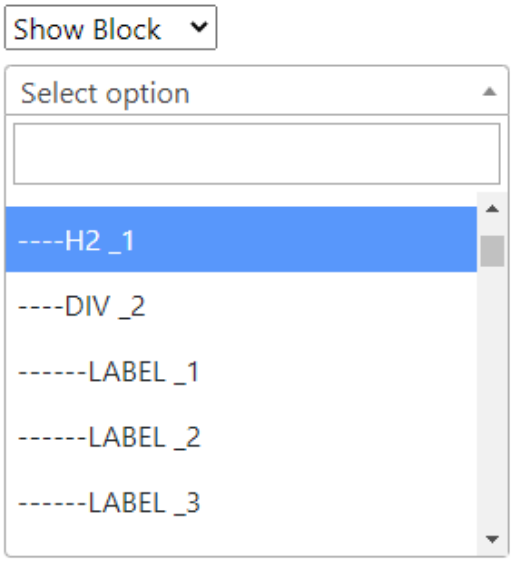


Ilustración 43 – Input bloque

- Mayor rango de reglas y filtrados. (expresiones regulares...)
- Disponer de un mayor rango de posibles inputs sobre los que trabajar. Actualmente, los input disponibles para crear reglas son text, radio, checkbox, url, tel, email, password, select y textarea. Por lo que, se mejoraría si se pudiera trabajar con fechas, colores o sliders entre otros.
- A la hora de crear un formulario se dan las opciones de crearlo mediante una URL o mediante insertando el formulario manualmente, una posible mejora sería en vez de tener que copiar y pegar el código HTML, realizar la inserción de código mediante la lectura de un fichero.
- Tener la opción de previsualizar el formulario a la hora de querer editarlo. Es decir, seleccionar el formulario que se desee y confirmar visualmente que se trata del formulario deseado antes de comenzar con la edición.
- En caso de que al usuario se le haya cerrado la sesión por inactividad mostrar un mensaje avisando de que la sesión se ha terminado.

Se considera que todas estas mejoras van más allá del ámbito inicial del proyecto y se proponen a la empresa cliente como posibles mejoras para una futura versión mejorada del software desarrollado, si se desease continuar con el proceso iterativo de mejoras.

Bibliografía

En este capítulo se referencian recursos y bibliografía que ha resultado de especial utilidad para la realización del proyecto y su memoria:

1. JavaScript https://uniwebsidad.com/libros/javascript?from=librosweb
2. NodeJs https://nodejs.org/en/about/
3. Libreria express-session https://www.npmjs.com/package/express-session#sessionoptions
4. Libreria i18n-express https://www.npmjs.com/package/i18n-express
5. Libreria html-validator https://www.npmjs.com/package/html-validator
6. MongoDB https://www.mongodb.com/es/what-is-mongodb
7. Visual Studio Code https://code.visualstudio.com/
8. https://es.wikipedia.org/wiki/Modelo_entidad-relaci%C3%B3n#Diagrama_entidad-relaci%C3%B3n
9. https://www.npmjs.com/package/mongoose
10. https://ergomedia.isl.gob.cl/app_ergo/nasatlx/

Anexo A Manual de usuario

En este capítulo se muestran algunos pantallazos de cómo es la aplicación de los usuarios y su interfaz.

A.1 - Manual para usuarios no registrados

Esta sección está destinada a aquellos usuarios que acceden por primera vez a la aplicación y no tienen cuenta. El objetivo de la siguiente guía es que conozcan el proceso de registro.

A.1.1 – Proceso de registro

El usuario no registrado accede a la herramienta y es dirigido a la página de identificación de usuario. En esta página se encuentra con dos utilidades por el momento; botones que le permiten cambiar de idioma (resaltado en tonalidad rojo) y un botón que redirige a la sección de registro (resaltado en tonalidad verde).

Ilustración 44 - Pantalla de identificación de usuario

A.1.2 – Registro de usuario

Si se ha seleccionado el botón de registrar, se muestra un formulario con los datos que se solicitan para crear un nuevo perfil.

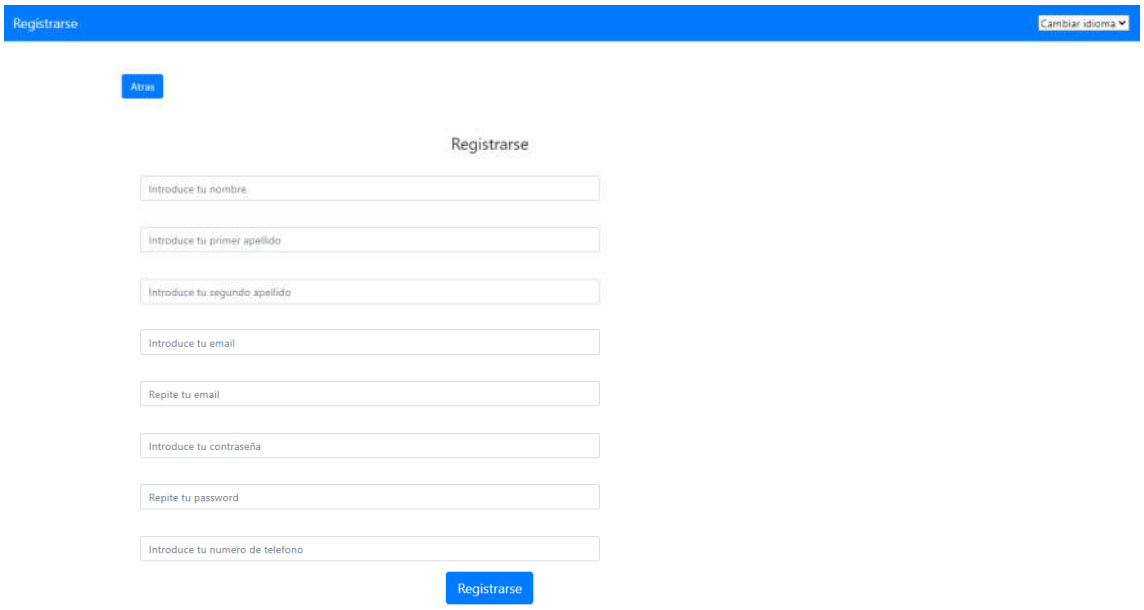


Ilustración 45 -Pantalla registrar usuario

Si falta alguno de los datos se mostrará una notificación en el campo faltante:

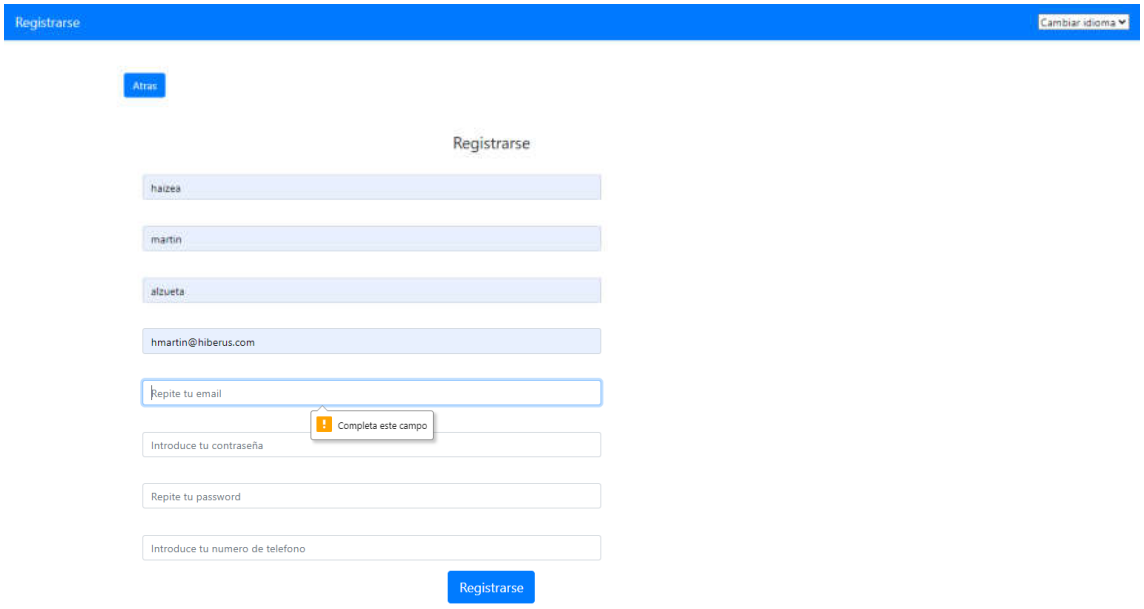


Ilustración 46 - Pantalla notificación en registrar usuario

Tras insertar los datos de forma correcta y al pulsar “Registrarse” pueden surgir varias situaciones:

- Todo está en orden y la creación de perfil ha sido satisfactoria, por lo que automáticamente nos redirige a la pantalla principal de la herramienta, la cual nos permitirá crear formularios.

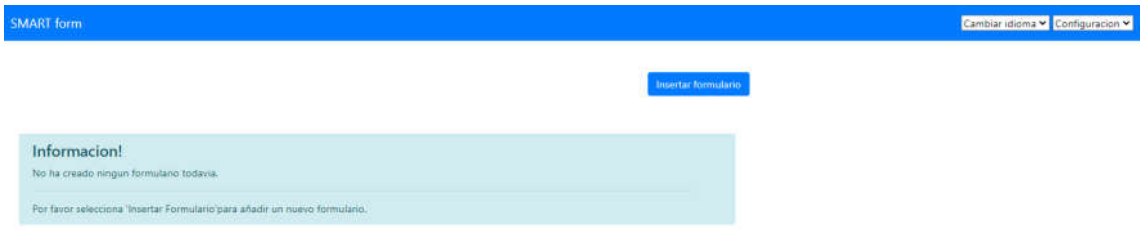


Ilustración 47 - Pantalla inicial una vez iniciado sesión

- La cuenta de usuario ya existe y se le comunica al usuario mediante el siguiente mensaje en pantalla.

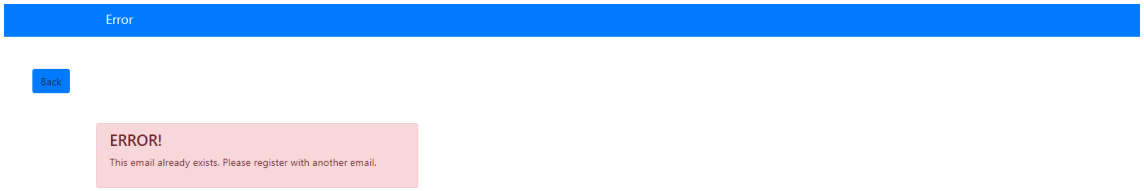


Ilustración 48 - Inicio de sesión erróneo

- Los emails o las contraseñas no coinciden, es decir, al introducir por segunda vez uno de los dos datos, no se ha introducido los mismos emails o contraseñas. Por lo tanto, nos mostrará un mensaje de error en pantalla indicándonos que datos no coinciden.

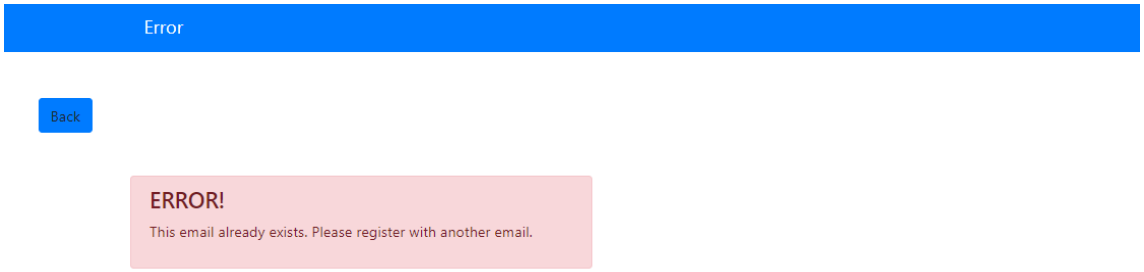


Ilustración 49 - inicio de sesión caso email errónea

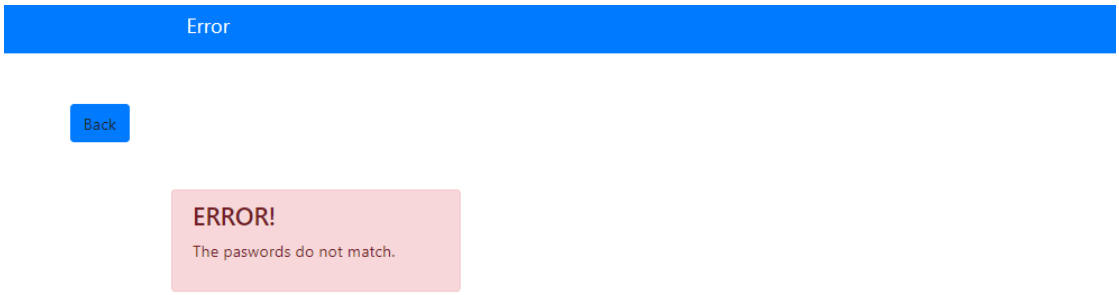


Ilustración 50 - inicio de sesión caso contraseña errónea

A.2 - Manual para usuarios registrados

En esta sección se explicará como un usuario, tras registrarse puede navegar por la herramienta. Esta ilustración representa la página inicial cuando un usuario ha iniciado sesión por primera vez.

Una vez creados formularios o haya iniciado sesión por segunda o más veces esta página será un listado con los formularios creados, el cual será comentado más adelante.

Ilustración 51 - Opciones disponibles para un usuario registrado

La secciones 1, 2 y 3 indicadas en la imagen están disponibles en todas las ventanas de la herramienta.

La sección 1 representa un botón para ir a la página inicial, página mostrada en la ilustración. Mientras que la sección 2 es para poder cambiar de idioma siempre que se desee.

La sección 3 es la opción de configuración que posee un usuario, el cual se comentará más adelante.

Finalmente, la sección 5, es un buscador sobre los formularios creados como se muestra en las siguientes ilustraciones:

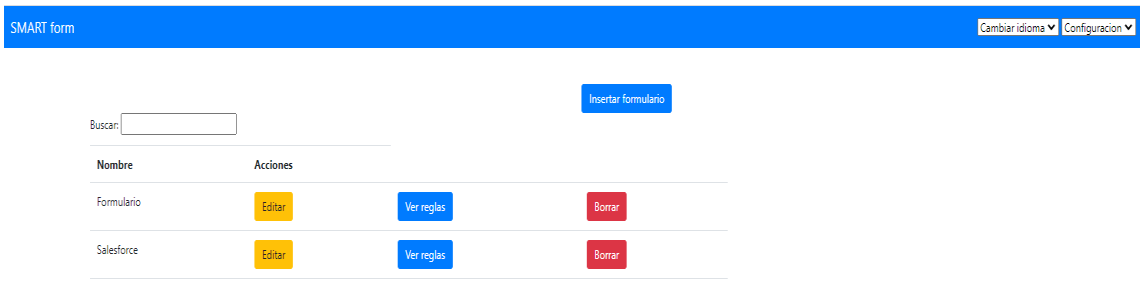


Ilustración 52 - Manual de usuarios buscador de formularios

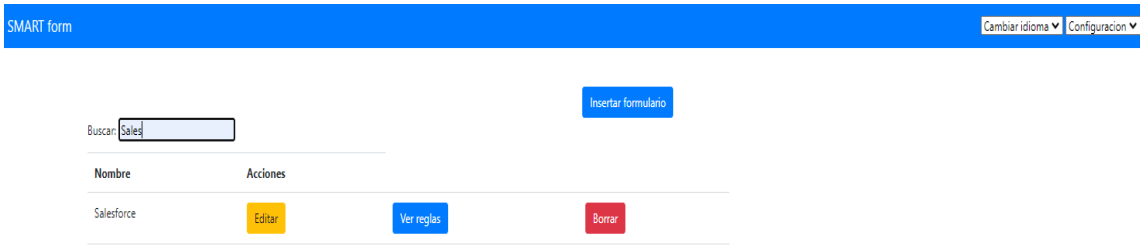


Ilustración 53 - Manual de usuarios funcionamiento del buscador de formularios

A.2.1- Configuración de usuario

En la sección 3 de la ilustración 38 representa la sección de configuración de usuario, esta sección estará disponible en todas las páginas de la herramienta.

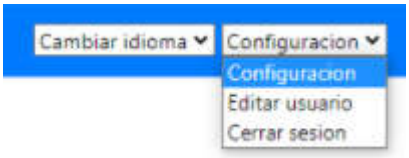


Ilustración 54 - Manual de usuarios sección configuración

Las opciones disponibles en la configuración son Editar Usuario y Cerrar sesión. Al editar el perfil del usuario únicamente podrá editar los campos: Nombre, Primer apellido, Segundo apellido y Teléfono, ya que el email lo utilizaremos como identificador del usuario.

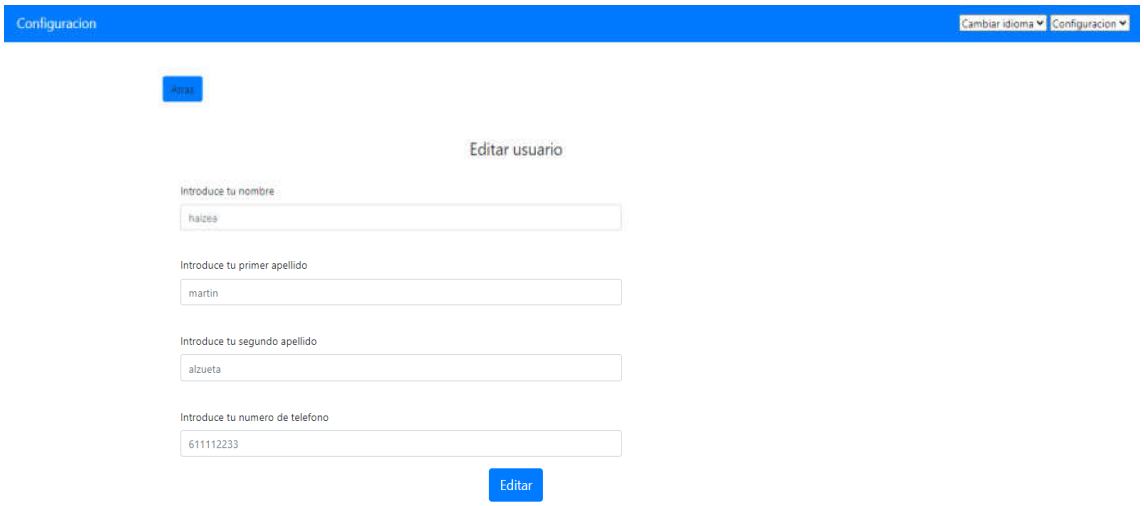


Ilustración 55 - Manual de usuarios editar usuario

A.2.2- Insertar formulario

Mediante la sección 4 de la ilustración 38 podremos crear un formulario, es decir, introduciremos el código HTML de un formulario creado con anterioridad o bien mediante una URL podremos acceder al código HTML deseado.

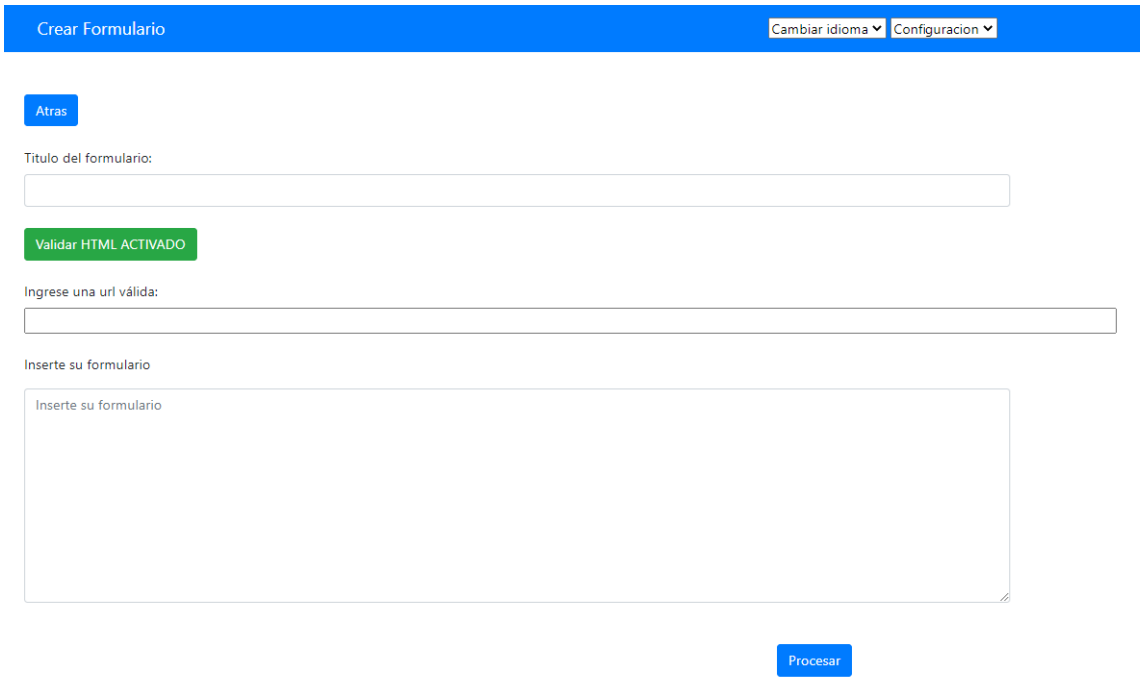


Ilustración 56 - Insertar formulario

Únicamente se introducirá por medio de una de las opciones URL o introduciendo el código puro. Una vez seleccionado una opción, la otra quedará inhabilitada como se muestra en las siguientes ilustraciones:

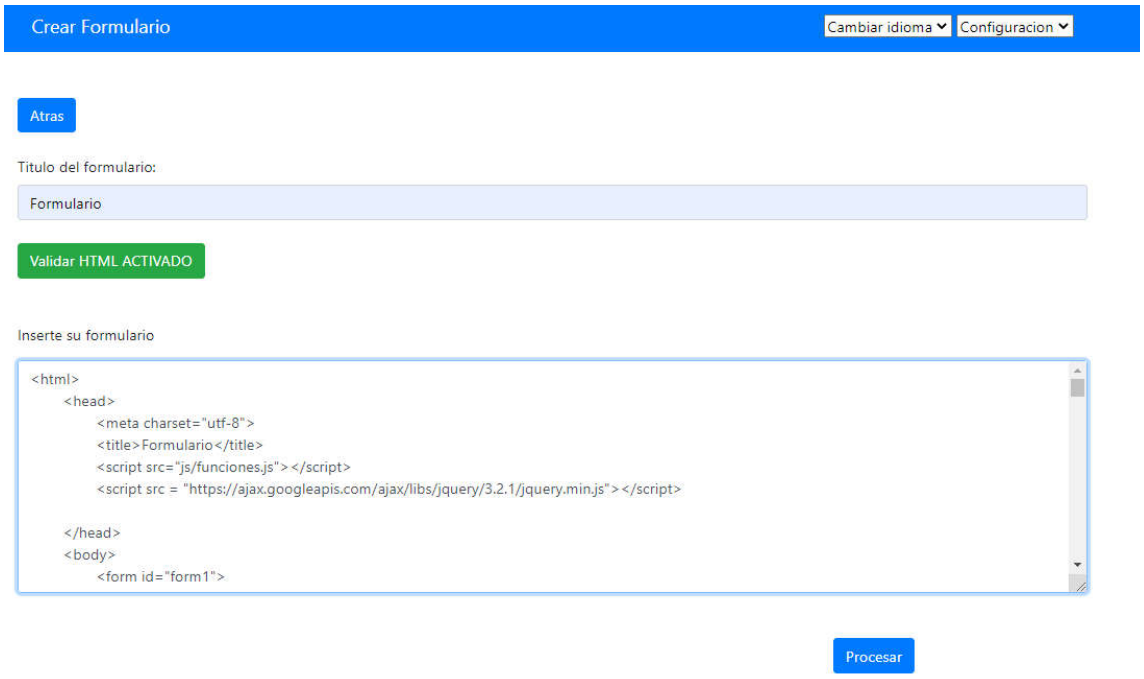


Ilustración 57- Insertar formulario por código HTML

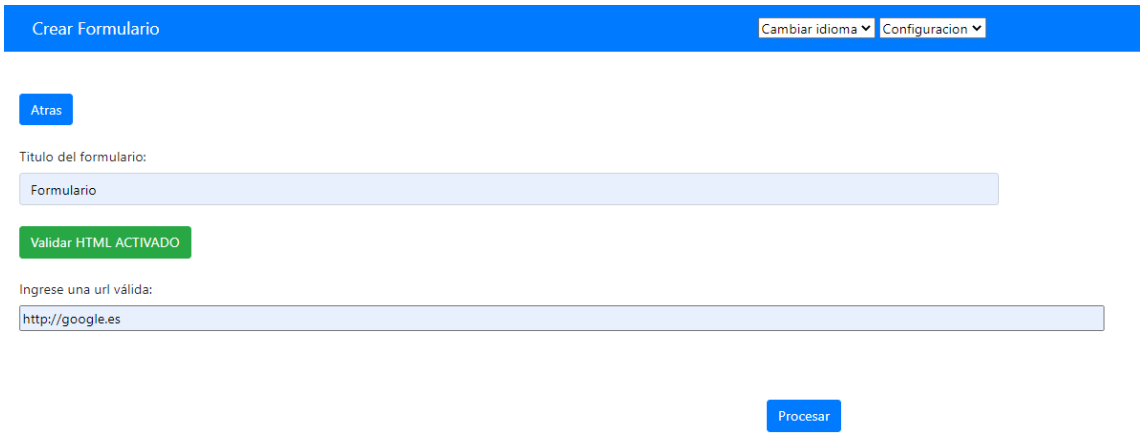


Ilustración 58 - insertar formulario por url

Por otro lado, se tendrá la opción de Activar o Desactivar la validación del código, el cual por defecto, realizará la validación.

Una vez procesado el código puede darse las siguientes situaciones:

- El código HTML contiene errores, por lo que el validador que hemos utilizado nos indicará mediante un mensaje de error los errores que contiene el HTML introducido.

Ilustración 59 - validación de formulario

- Se encuentran más de un formulario en un mismo código HTML. Por lo que, se seleccionará mediante un selector el formulario sobre al que se le aplicarán reglas.

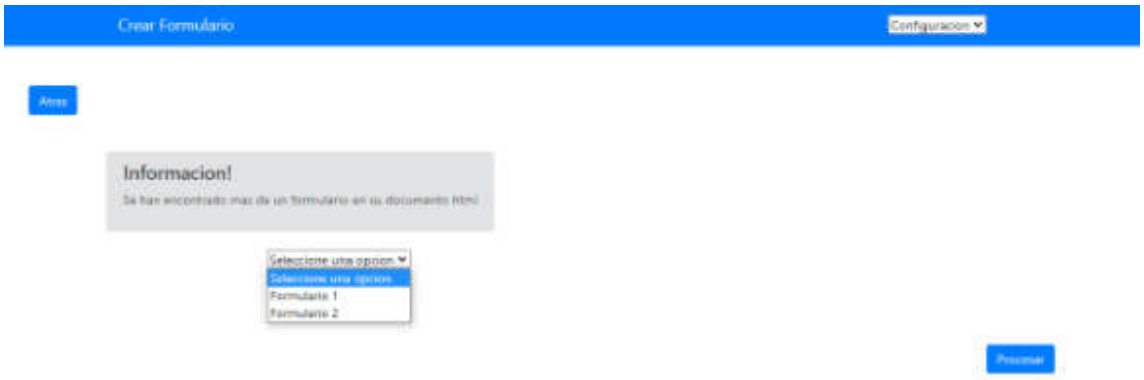


Ilustración 60 - seleccionar el formulario deseado

Ya que puede que no se sepa qué formulario es con el que se quiere trabajar a la hora de elegir una opción aparecerá una pre-visualización del formulario para decidir mejor cual seleccionar.

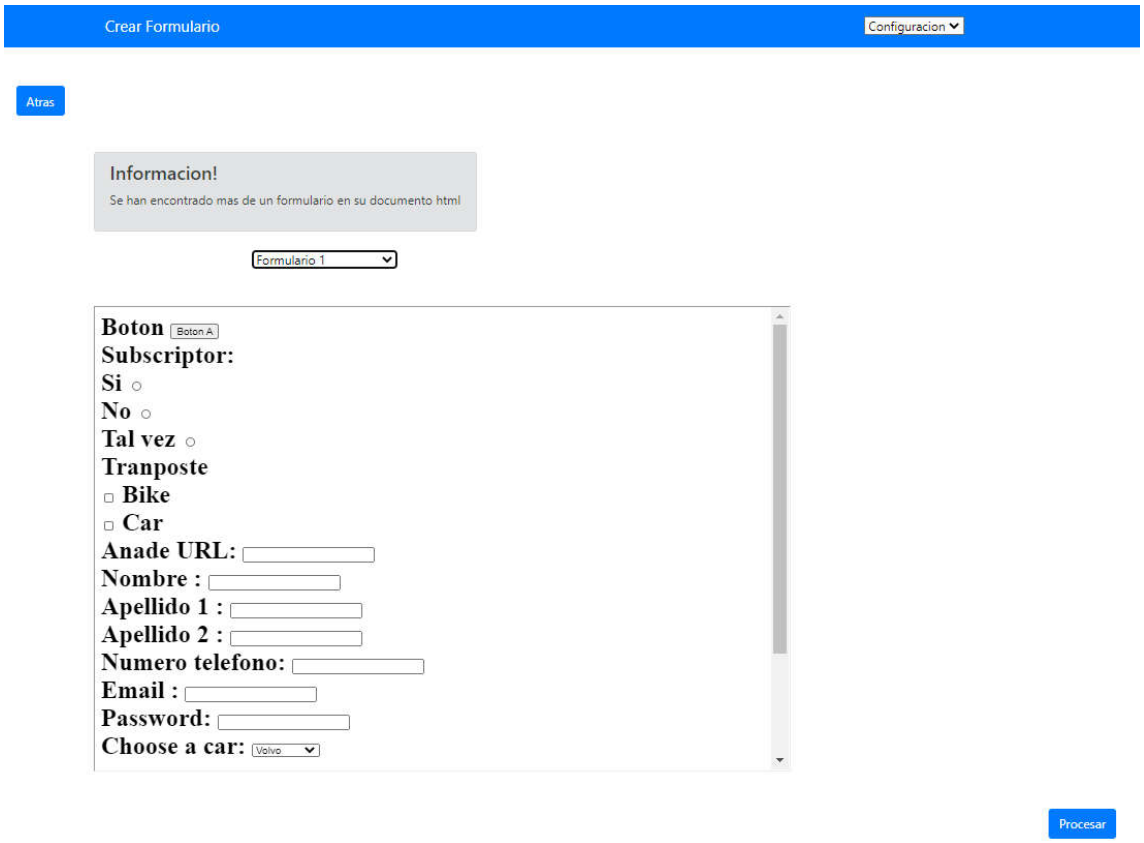


Ilustración 61 - pre visualización del formulario

Una vez procesado, llegaríamos a la pantalla la cual mostrará el listado de reglas y por lo tanto se podría comenzar con la creación de reglas.

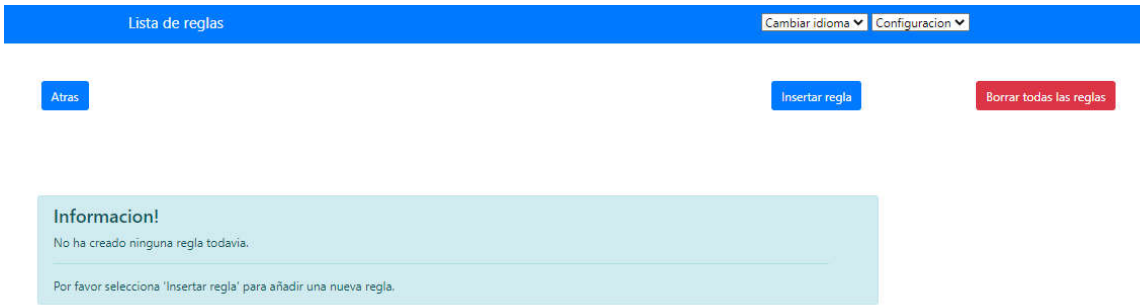


Ilustración 62 - listado de reglas

A.2.3- Editar formulario

Una vez se hayan creado formularios se tendrá la opción de Editar, Ver reglas y Borrar formulario como se muestra en la ilustración. Para editar un formulario seleccionaremos la opción de Editar (resaltado en rojo).

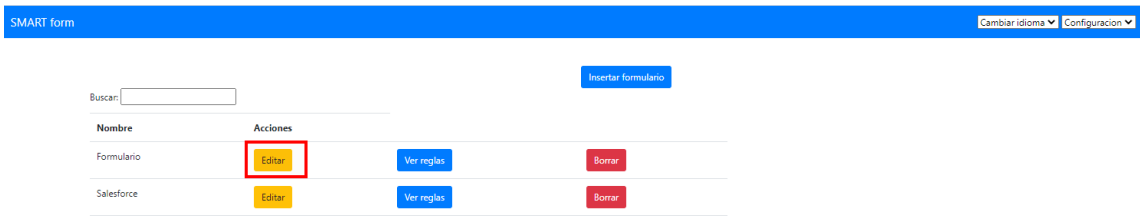


Ilustración 63 - editar formulario

Pueden darse dos situaciones una vez que se quiera editar un formulario.

- El formulario no tiene ninguna regla creada por lo que permitirá editarlo sin problemas donde también se tendrá la opción de validar el HTML.

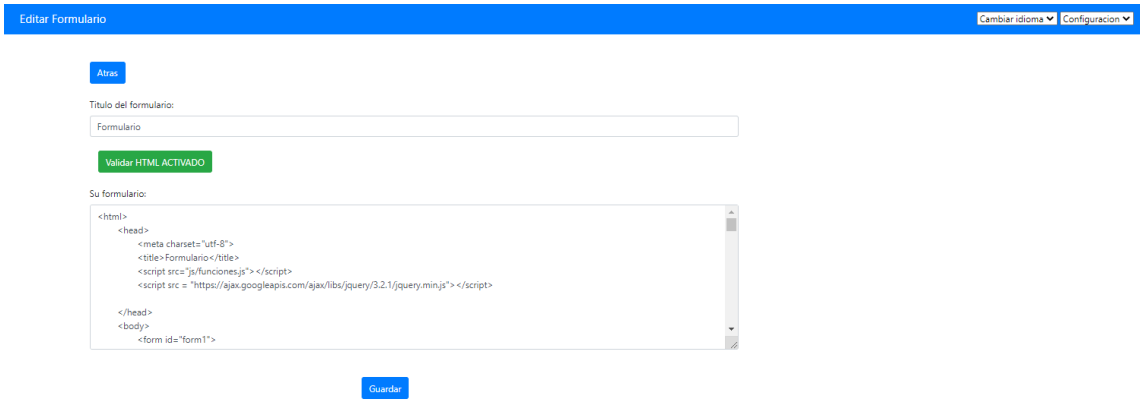


Ilustración 64 - opciones de editar formulario

- El formulario tiene reglas creadas por lo que no permitirá editar dicho formulario hasta que no se eliminen todas las reglas. Si se encuentra en este caso se mostrará una ventana de información indicando que hay reglas creadas.

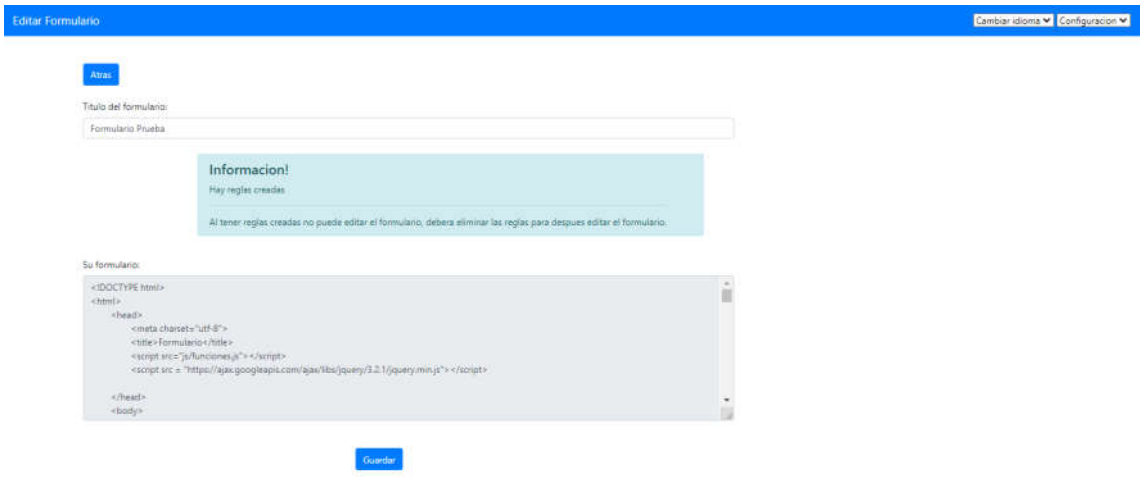


Ilustración 65 - editar formulario con reglas creadas

A.2.4- Borrar formulario

Otras de las opciones es poder eliminar el formulario, el cual eliminará de la base de datos toda información asociada, reglas creadas, input encontrados…

Para poder borrar el formulario se selecciona la opción de borrar (resaltado en rojo en la ilustración).

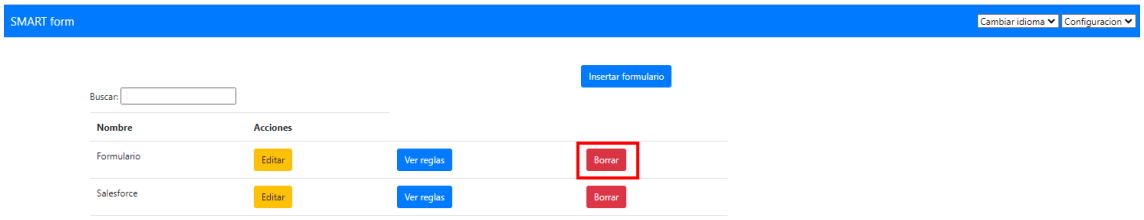


Ilustración 66 - opción borrar formulario

Una vez seleccionado la opción, se nos mostrará los datos del formulario, el cual se utilizara para saber si se está seguro de querer eliminar dicho formulario. Finalmente, volviendo a seleccionar Borrar, se tendrá el formulario con todos sus datos asociados eliminados.

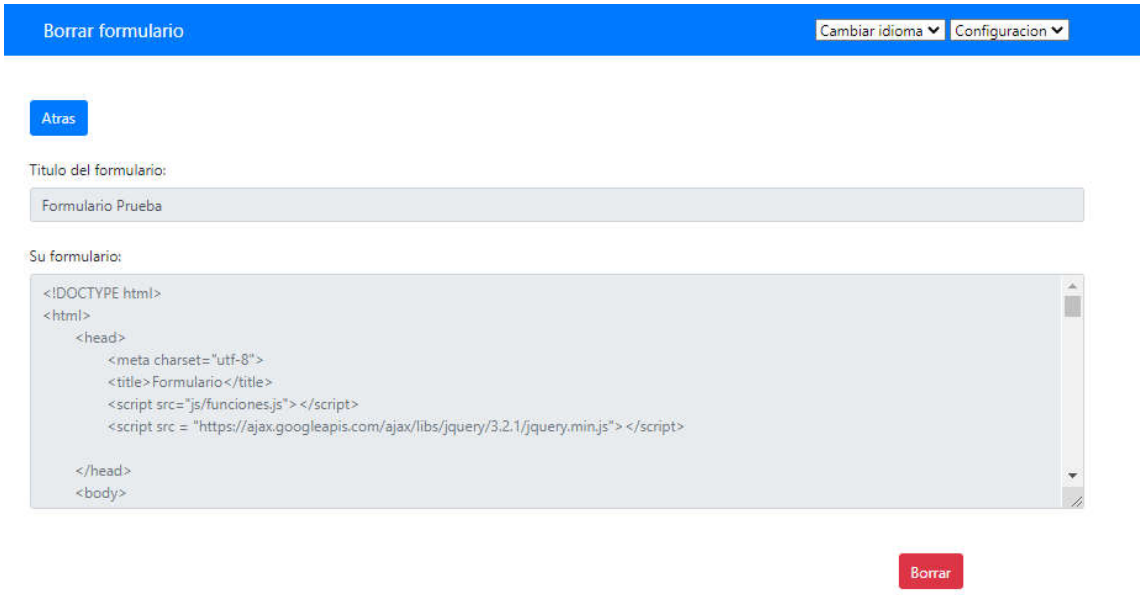


Ilustración 67 - validación para borrar formulario

A.2.5- Insertar regla

Para insertar una regla bien se puede hacer después de haber insertado un formulario o mediante la opción de Ver reglas de un formulario determinado.

Para insertar una nueva regla se deberá seleccionar la opción insertar regla (resaltado en rojo en la ilustración).

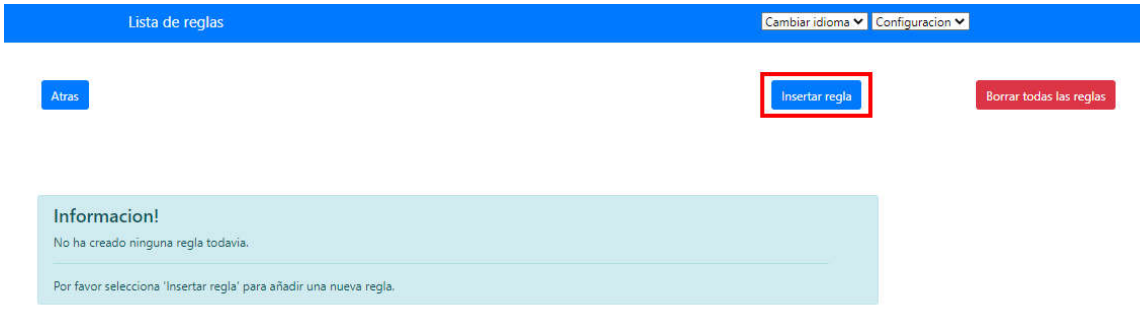


Ilustración 68 - insertar regla

La estructura de una regla será la siguiente: Si condicionante entonces determinante. Un ejemplo sencillo sería: Si el input con el id nombre no está vacío entonces mostramos el input con el id ap1.

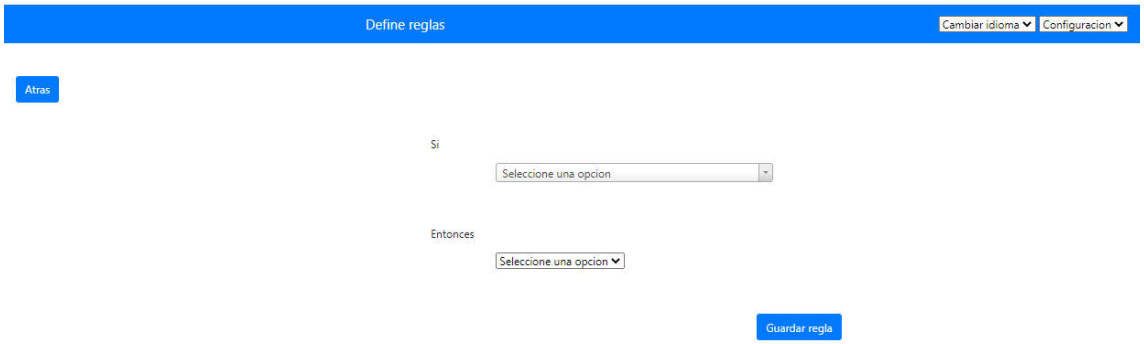


Ilustración 69- definir regla

Los condicionantes disponibles son los que se muestran en la siguiente imagen.

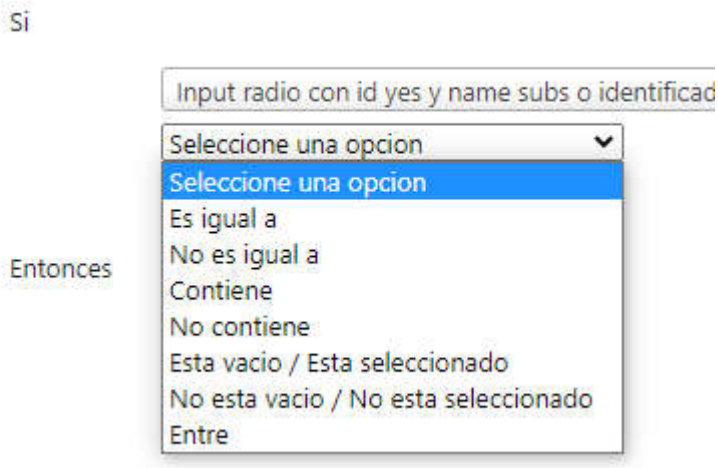


Ilustración 70 - condiciones disponibles para creación de reglas

Dependiendo del condicionante seleccionado se irán mostrando más opciones, por ejemplo, si seleccionamos el condicionante Entre tendríamos la opción de elegir entre que valores numéricos tendría que estar.

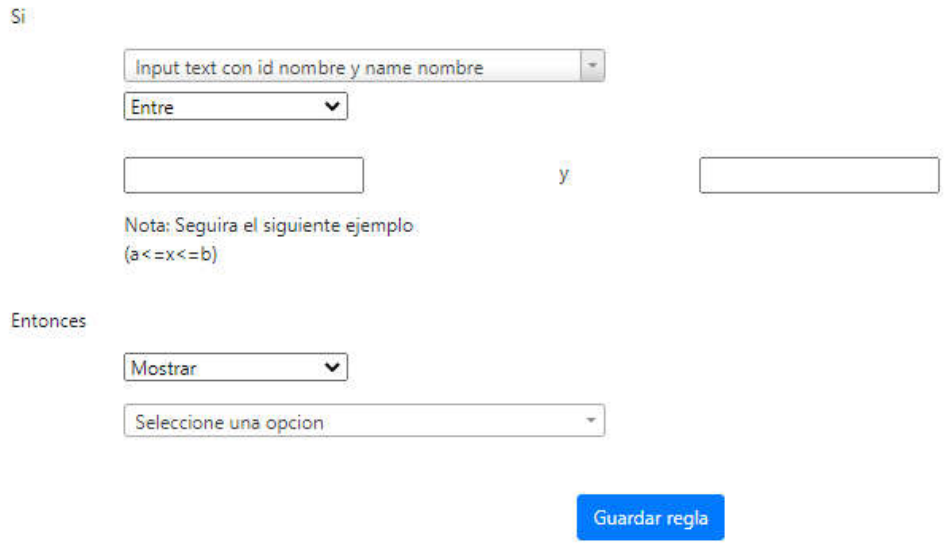


Ilustración 71 - ejemplo de crear una regla

Respecto al determinante las opciones disponibles son Mostrar, no mostrar, habilitar un campo, hacer un campo obligatorio o mostrar bloque.

Destacaría la opción Mostrar bloque, ya que esta opción es útil cuando se quieren mostrar un conjunto de campos a la vez. Esta opción se utiliza sobre todo cuando hay campos que no tienen identificadores o nombres aplicados por lo que se utilizará para poder identificarlos y aplicarles reglas.

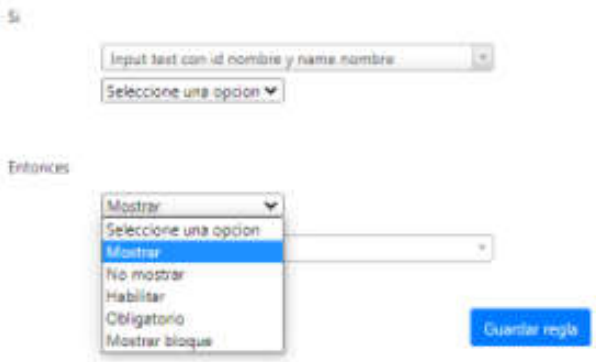


Ilustración 72 - condicionales disponibles

Finalmente, hay que destacar que los selectores de campos tienen un buscador integrado.

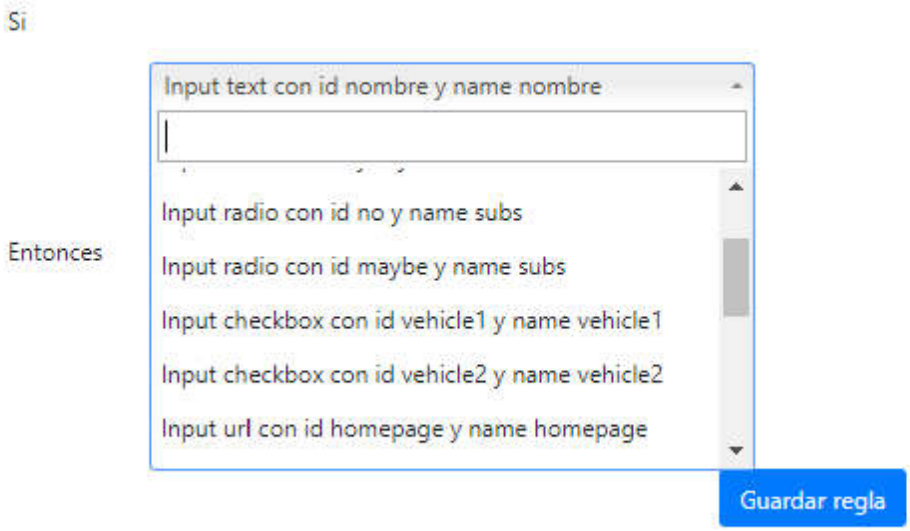


Ilustración 73 - selectores de input para definir reglas

A.2.6- Ver reglas

La opción de “Ver Reglas” nos mostrará un listado con las reglas creadas hasta el momento además de darnos la opción de crear nuevas reglas o borrar todas las reglas. Y a su vez, cada regla tendrá la opción de ser editada o borrada.

Por otro lado, si hay reglas generadas tendremos la opción de generar formulario dinámico.

Para poder ver las reglas de un formulario se ha de seleccionar la opción de “Ver reglas” (resaltado en rojo en la ilustración).

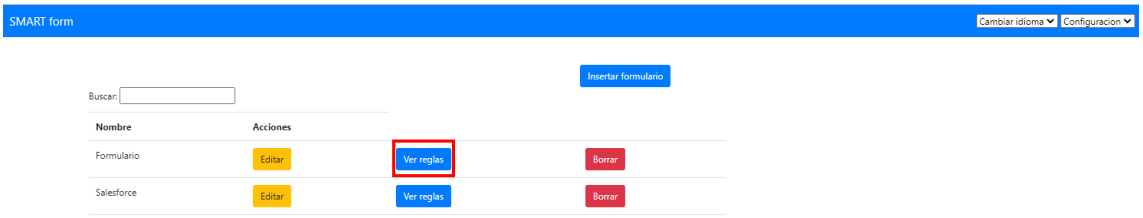


Ilustración 74 - visualización de reglas

Como se ha comentado anteriormente, nos mostrará un listado de las reglas creadas como se muestra en la imagen.

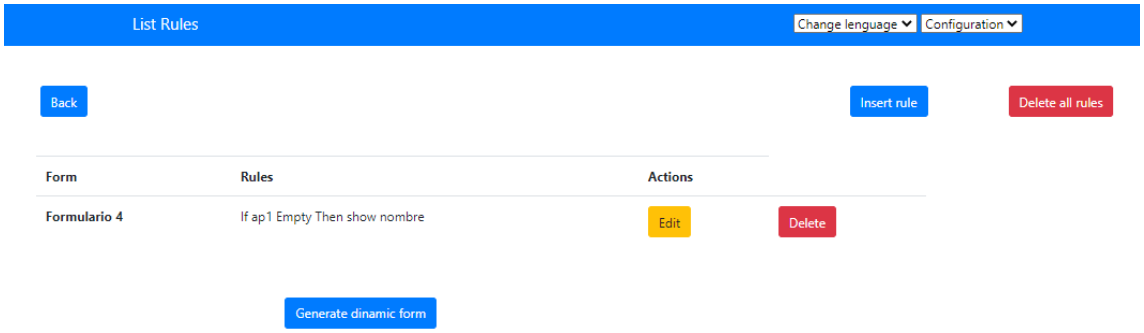


Ilustración 75 - listado de reglas

A.2.7- Editar regla

La opción de editar regla estará habilitada siempre y cuando se haya creado una regla con anterioridad (opción resaltada en rojo en la ilustración).

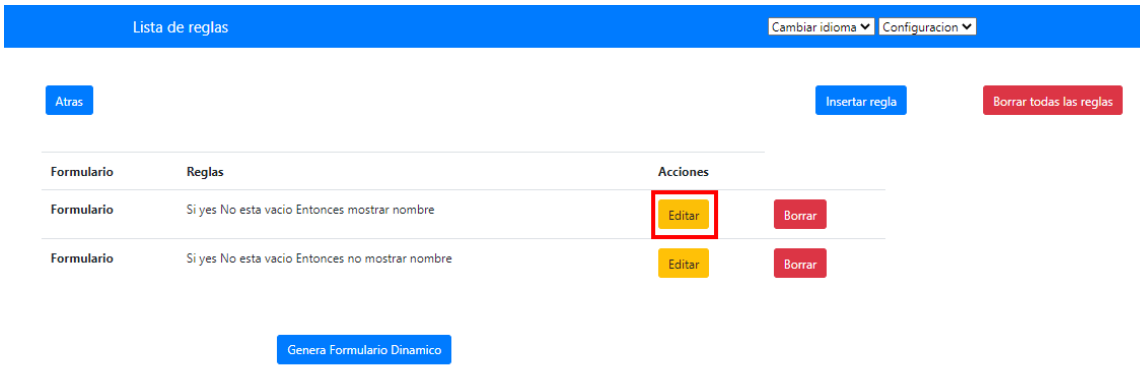


Ilustración 76 - editar reglas

La estructura para editar una regla es la misma que a la hora de crearla.

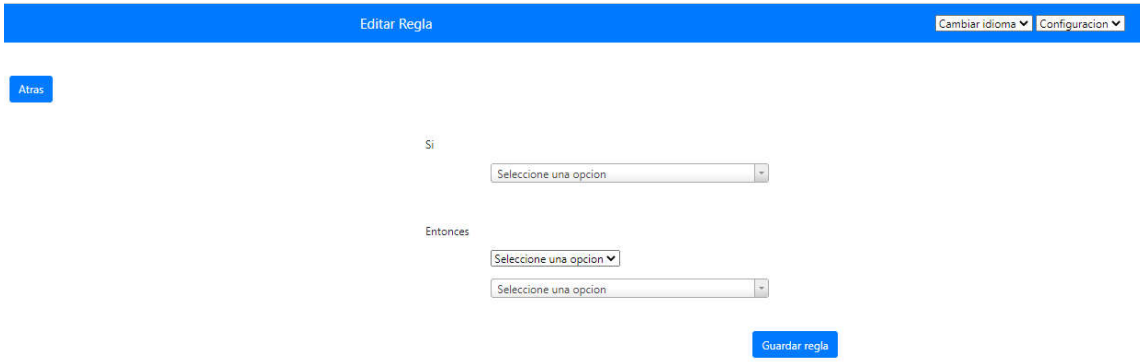


Ilustración 77 - definición de editar regla

Por lo que el proceso de edición será el mismo que el de creación de reglas visto en el apartado A.2.5-Insertar regla .

A.2.8- Borrar regla

La opción de borrar regla estará habilitada siempre y cuando se haya creado una regla con anterioridad (opción resaltada en rojo en la ilustración 78).

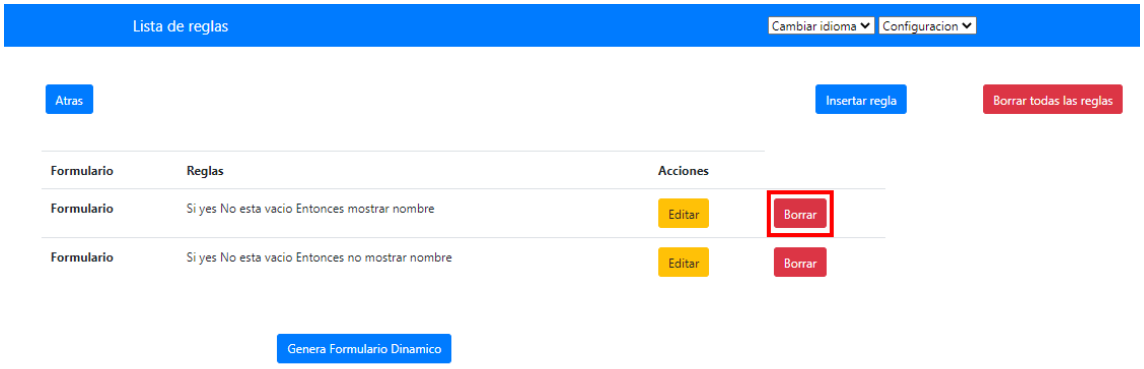


Ilustración 78 - opción borrar regla

Una vez seleccionada dicha opción, visualizamos la regla de manera que servirá para validar si realmente se quiere eliminar la regla seleccionada, ya que una vez eliminada no se podrá recuperar a no ser que se vuelva a crear la regla.

Ilustración 79 - validación borrar regla

A.2.9 Borrar todas las reglas

La opción Borrar todas las reglas, como su nombre indica, eliminará todas las reglas relacionadas con el formulario seleccionado (opción resaltada en rojo en la ilustración).

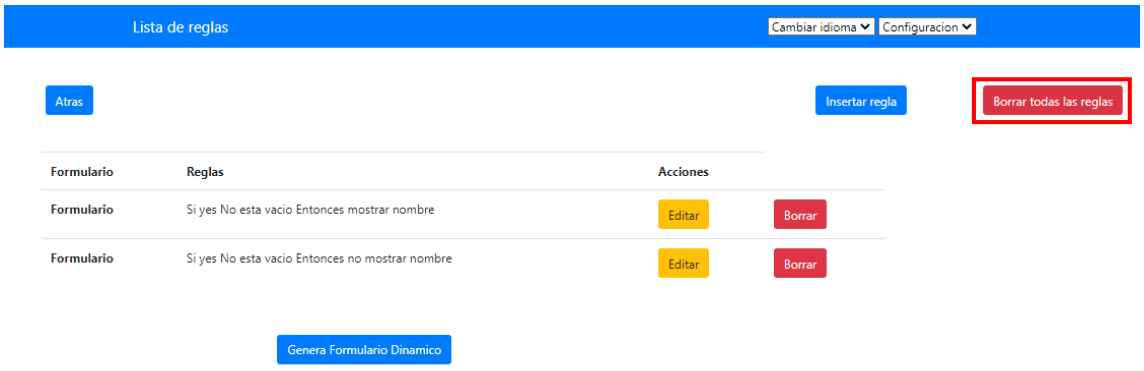


Ilustración 80- opción borrar toda las reglas

Una vez seleccionada nos mostrará un mensaje de que se está queriendo eliminar todas las reglas, el cual sirve como validador de que el usuario está queriendo eliminarlas.

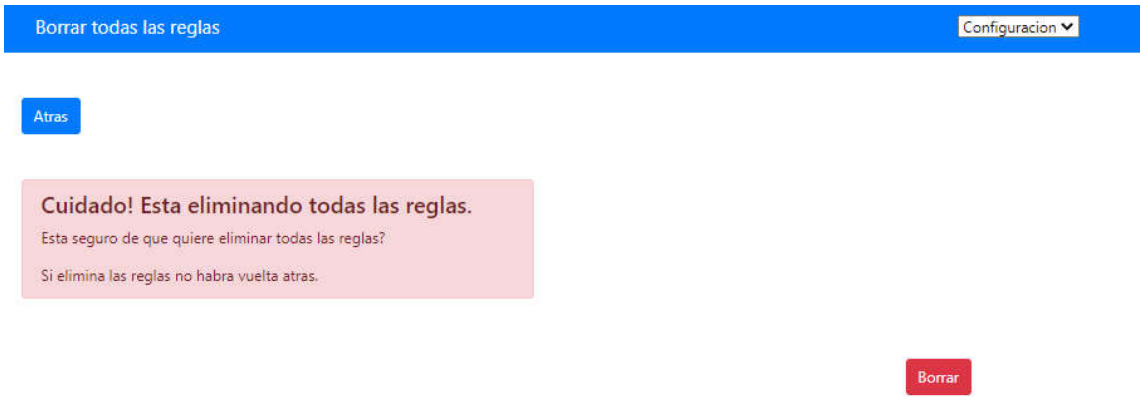


Ilustración 81- validación borrar todas las reglas

A.2.10 Generar formulario dinámico

La opción de “Generar formulario dinámico” estará habilitada siempre y cuando se haya creado una regla con anterioridad (opción resaltada en rojo en la ilustración 82).

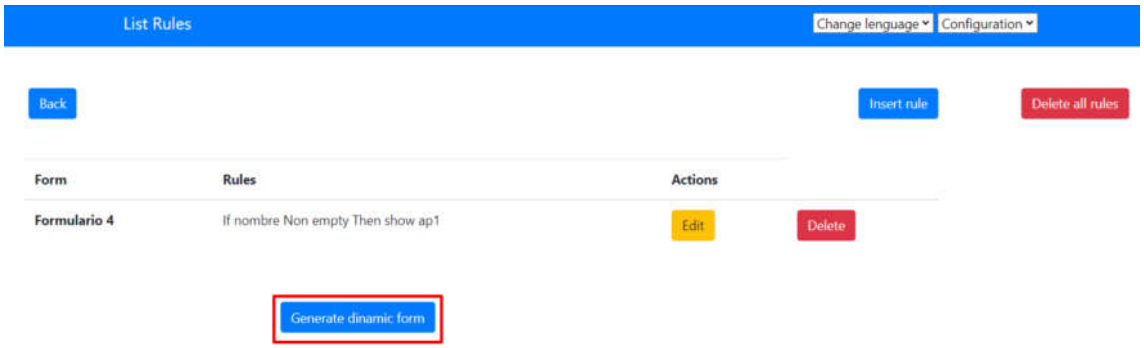


Ilustración 82- opción generar formulario dinámico

Una vez seleccionado la opción de “Generar formulario dinámico” se generará un código JavaScript que reúna todas las reglas configuradas visualizado en la sección 1 resaltada en rojo de la siguiente imagen. Por otro lado, si no se desea copiar y pegar el código se tiene la opción de descargar el código generado en formato JS o en formato HTML (secciones 2 y 3 resaltadas en rojo respectivamente).

Ilustración 83 - descargar formulario dinámico

Anexo B Pruebas de usuario

En este capítulo se aportará la documentación utilizada a la hora de realizar pruebas a usuarios.

B.1 - Formulario

A continuación se mostrará el formulario que los participantes han utilizado para realizar pruebas en la herramienta y una captura de su visualización.

```
<html>
<head>
        <meta charset="utf-8">
        <title>Formulario</title>
        <script src =
"https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></
script>
        <script src="js/funciones.js"></script>
</head>
<body>
        <form id="formulario1">
                <div class="form1">
                        <h2 align="center">Formulario</h2>
                        <div>
                                <label >Subscriptor: </label><br>
                                <label>Si
                                <input type="radio" id="yes"
name="subs" value="yes" ></label><br>
                                <label>No
                                <input type="radio" id="no"
name="subs" value="no" ></label><br>
                        </div>
                        <div>
                         <label for="nombre">Nombre : </label>
                        <input type="text" id="nombre" name="nombre">
                    </div>
                        <div>
                         <label for="ap1">Apellido 1 : </label>
                        <input type="text" id="ap1" name="ap1">
                    </div>
                    <div>
                         <label for="ap2">Apellido 2 : </label>
                        <input type="text" id="ap2" name="ap2">
                    </div>
                        <div>
                     <label for="bornDate"> Edad
                     <input type="number" id="edad"  name="edad"  >
</label>
                </div>
                        <div id="telG"  class="telGCss">
                                <label for="phone">Numero
```

```
telefono:</label>
                                <input type="tel" id="phone"
name="phone">
                        </div>

                        <div >
                                <label for="email">Direccion
email</label>
                                <input type="email" id="email" />
                        </div>
                        <div >
                                <label  for="pass">Contraseña</label>
                                <input type="password" id="pass"  />
                        </div>
                        <div>
                                <label  for="repite_pass">Repite
Contraseña</label>
                                <input type="password"
id="repite_pass" class="form-control" />
                        </div>
                        <div >
                                <label id="pass_no_coincide"
for="Diferente_pass">La contraseña no coincide</label> <br>
                        </div>
                        <div >
                                <label id="pass_coincide"
for="Igual_pass">La contraseña  coincide</label> <br>
                        </div>
                        <div >
                                <label for="interest">¿Que tipo de
cosas te interesan más?: </label><br>
                                <label for="Deportes">
Deportes</label>
                                <input type="checkbox" id="Deportes"
name="Deportes" value="Deportes"><br>
                                <label for="Futbol"> Futbol</label>
                                <input type="checkbox" id="Futbol"
name="Futbol" value="Futbol"><br>
                                <label for="Videojuegos">
Videojuegos</label>
                                <input type="checkbox"
id="Videojuegos" name="Videojuegos" value="Videojuegos"><br>
                                <label for="Nacional">
Nacional</label>
                                <input type="checkbox" id="Nacional"
name="Nacional" value="Nacional"><br>
                                <label for="Economía">
Economía</label>
                                <input type="checkbox" id="Economía"
name="Economía" value="Economía"><br>
                                <label for="Cultura"> Cultura</label>
                                <input type="checkbox" id="Cultura"
name="Cultura" value="Cultura"><br>
                                <label for="Política">
Política</label>
                                <input type="checkbox" id="Política"
name="Política" value="Política"><br>
                                <label for="Internacional">
Internacional</label>
                                <input type="checkbox"
```

```
id="Internacional" name="Internacional" value="Internacional"><br>
                                <label for="Otro"> Otros</label>
                                <input type="checkbox" id="Otro"
name="Otro" value="Otro"><br>
                                <textarea id="w3review"
name="OtrosValores" class="OtrosValores" rows="3" cols="100"
placeholder="Escribe con que otras cosas te
interesan"></textarea><br>
                        </div>

                        <button type="submit" class="btn btn-primary
btn-block mb-4">Enviar</button>
                </div>
        </form>
        <form>
                <div class="form2">
                        <div>
                                <label
for="opiniones">Opiniones:</label>
                                <textarea id="opiniones"
name="opiniones" class="opiniones" rows="3" cols="100"
placeholder="Escriba su opinion"></textarea>
                        </div>

                        <div>
                                <label> Valoracion de la herramienta
(0 bajo - 10 alto)</label><br>
                                <label><input type="radio" id="val0"
name="val" value="0" >0</label>
                                <label><input type="radio" id="val1"
name="val" value="1" >1</label>
                                <label><input type="radio" id="val2"
name="val" value="2" >2</label>
                                <label><input type="radio" id="val3"
name="val" value="3" >3</label>
                                <label><input type="radio" id="val4"
name="val" value="4" >4</label>
                                <label><input type="radio" id="val5"
name="val" value="5" >5</label>
                                <label><input type="radio" id="val6"
name="val" value="6" >6</label>
                                <label><input type="radio" id="val7"
name="val" value="7" >7</label>
                                <label><input type="radio" id="val8"
name="val" value="8" >8</label>
                                <label><input type="radio" id="val9"
name="val" value="9" >9</label>
                                <label><input type="radio" id="val10"
name="val" value="10" >10</label>
                        </div>
                </div>
        </form>

</body>
<style type="text/css">
.form1 {
        background-color: #fafafa;
        padding: 1rem;
        height: 100%;
        width: 50%;
```

```
        margin: auto;
        margin-bottom: 50px;
        font-size: 30px;
        /* IMPORTANTE */
        text-align: left;
}
.form2 {
        background-color: #fafafa;
        padding: 1rem;
        height: 100%;
        width: 50%;
        margin: auto;
        font-size: 30px;
        /* IMPORTANTE */
        text-align: left;
}
input{
        height:25px
}
</style>
</html>
```

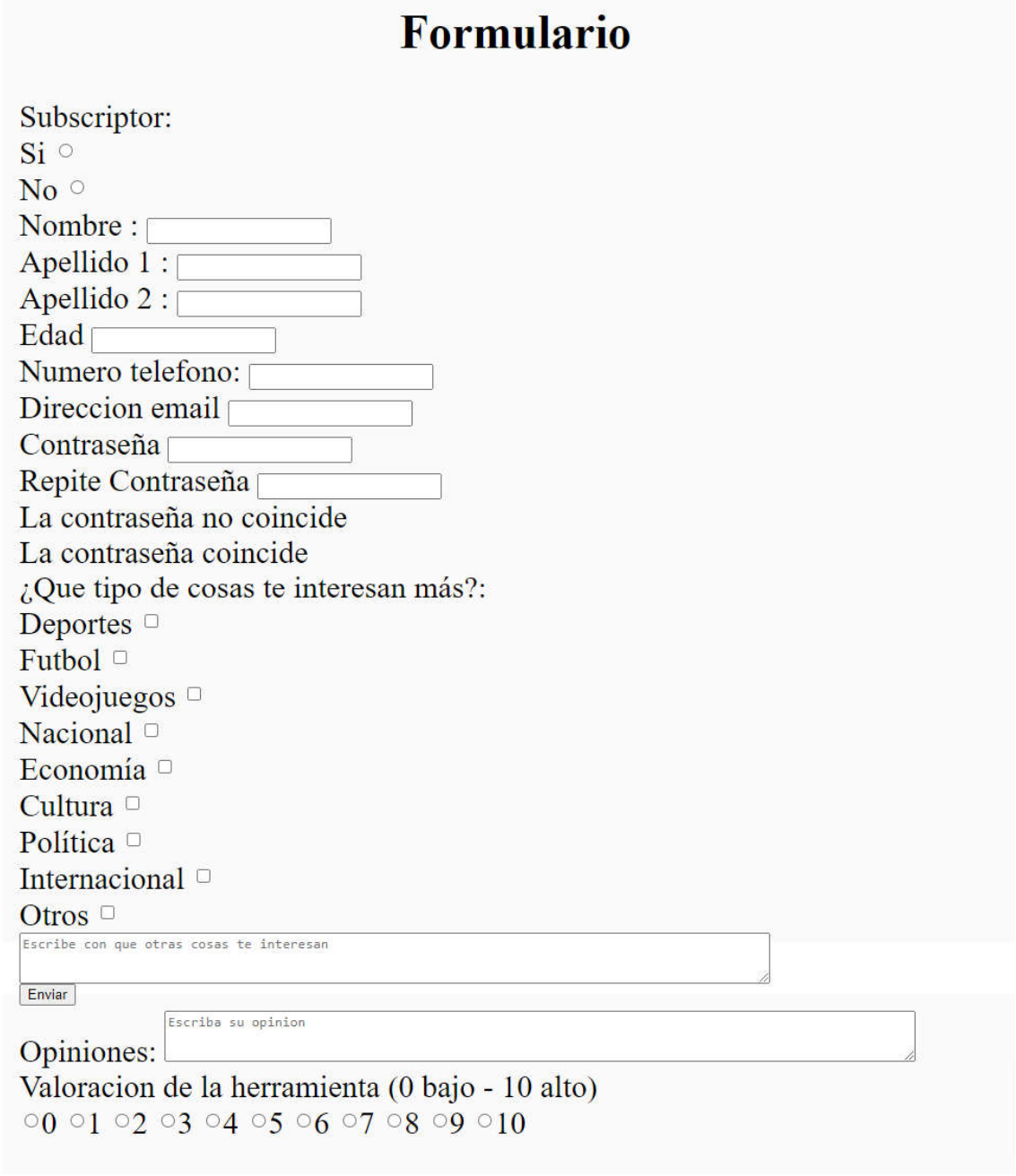


Ilustración 84- formulario de ejemplo

B.2 - Documento entregado a los participantes

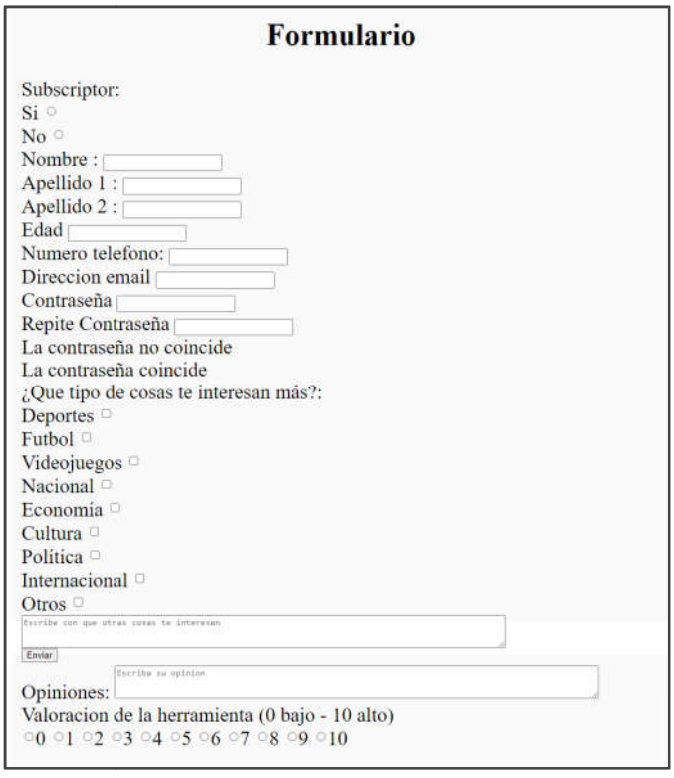
A continuación se mostrará el documento que se ha entregado a los participantes para la realización de pruebas de usuario.

Nombre: _____ Edad: _____ Género: _____ Fecha: _____

Conocimientos de desarrollo web: ☐Cero ☐Avanzado

Introducción

- Vamos a dedicar aproximadamente 40 minutos a realizar DOS RETOS relacionados con el desarrollo web HTML y JavaScript.
- Cada uno de estos retos se compone de hasta 6 tareas. Lee todo el documento, asegúrate de comprender lo que se pide en cada tarea, pregunta a la persona que conduce el test si tienes alguna duda.
- No comiences a desarrollar una tarea sin haber finalizado **o dado por abandonada** la anterior.
- Avisa a la persona que conduce el test cuando das por finalizada/abandonada una tarea y pasas a la siguiente.
- Si realizar una tarea te lleva más de 5 minutos se te invitará a darla por abandonada y continuar con la siguiente.
- Puedes utilizar internet para realizar cualquier consulta que consideres pueda ayudarte a cumplir los objetivos.
- Vas a trabajar con un formulario HTML como el siguiente.



☐ Estoy realizando este reto A en primer lugar.

Nombre: _____ ☐ Ya he realizado el reto A anteriormente.

A.1) La persona que conduce el experimento te va a dar unas indicaciones básicas sobre cómo atajar el reto.

☐ Voy a desarrollar escribiendo yo mism@ código javascript con un editor de código

☐ Voy a desarrollar mediante una herramienta/interfaz de ayuda al desarrollo.

A.2) Realiza las siguientes tareas, una a una, en orden, y avisando a la persona que conduce la prueba cada vez que termines/comiences una nueva tarea.

Modelar los siguientes comportamientos.

A.2.1) Avisa a la responsable e intenta que: Si el usuario indica que no está suscrito, mostrar el campo "Nombre" y viceversa.

A.2.2) Avisa a la responsable e intenta que: Si el usuario indica que no está suscrito, hacer obligatoria el rellenar el campo "Nombre".En cambio, si está suscrito no debe ser obligatorio rellenar el campo "Nombre"

A.2.3) Avisa a la responsable e intenta que: Si las 2 contraseñas introducidas por el usuario son diferentes, mostrar el bloque que indica "La contraseña no coincide", de lo contrario ocultarlo.

A.2.4) Avisa a la responsable e intenta que: Si las 2 contraseñas introducidas por el usuario son iguales, mostrar el bloque que indica "La contraseña coincide", de lo contrario ocultarlo.

A.2.5) Avisa a la responsable e intenta que: Si el usuario marca el interés "Deportes", mostrar el bloque de input checkbox "Fútbol". De lo contrario no se debe mostrar.

A.2.6) Avisa a la responsable e intenta que: Si edad está entre 18 y 65 habilitar el campo para introducir número de teléfono. De lo contrario deshabilitarlo.

Has finalizado esta fase. Por favor, avisa a la persona que conduce la prueba y rellena el formulario presente al reverso de este mismo folio.

Exigencia mental: ¿Qué tan demandante mentalmente es el reto? (seleccione una opción, 0 bajo, 10 alto)

□0 □1 □2 □3 □4 □5 □6 □7 □8 □9 □10

Exigencia temporal: ¿Qué tan fuerte o rápido es el ritmo impuesto para hacer el reto? (seleccione una opción, 0 bajo, 10 alto)

□0 □1 □2 □3 □4 □5 □6 □7 □8 □9 □10

Esfuerzo: ¿Que tan duro tiene que trabajar para lograr un adecuado nivel de rendimiento? (seleccione una opción, 0 bajo, 10 alto)

□0 □1 □2 □3 □4 □5 □6 □7 □8 □9 □10

Rendimiento: ¿Qué tan exitoso ha sido lograr el reto propuesto? (seleccione una opción, 0 bajo, 10 alto)

□0 □1 □2 □3 □4 □5 □6 □7 □8 □9 □10

Nivel de frustración: ¿Qué tan inseguro, irritado o estresado y molesto se ha encontrado a la hora de realizar el reto? (seleccione una opción, 0 bajo, 10 alto)

□0 □1 □2 □3 □4 □5 □6 □7 □8 □9 □10

Extra:

Comenta con la persona que ha conducido la prueba cualquier detalle, sugerencia u observación sobre los retos o su desarrollo.

B.3 - Resultados individuales

A continuación se mostraran los resultados individuales y anonimizados obtenidos en las pruebas de usuario.

En la siguiente captura se mostraran los resultados obtenidos en el caso de realizar la prueba sin la herramienta:

				JAVASCRIPT PURO																		
				NASA-TLX					TASK													
Participante	sexo	tipo	comienza c	1	2	3	4	5	Tiempo T1	Exito T1	Tiempo T2	Exito T2	Tiempo T3	Exito T3	Tiempo T4	Exito T4	Tiempo T5	Exito T5	Tiempo T6	Exito T6	Total éxitos	Total fallos	Consultas a documentacion
USER1	Hombre	experto	heramienta	10	10	8	2	9	600	fallo	350	Exito	600	fallo	120	fallo	180	exito	120	exito	3	3	18
USER2	Hombre	experto	JS	8	9	8	1	9	600	fallo	600	fallo	600	fallo	600	fallo	0	fallo	0	fallo	0	6	17
USER3	Mujer	experto	JS	7	6	9	3	9	600	fallo	600	fallo	600	fallo	300	fallo	300	fallo	300	fallo	0	6	8
USER4	Mujer	novato	heramienta	10	10	8	0	10	600	fallo	600	fallo	600	fallo	600	fallo	600	fallo	600	fallo	0	6	0
USER5	Hombre	novato	heramienta	10	10	10	1	10	600	fallo	600	fallo	600	fallo	600	fallo	600	fallo	600	fallo	0	6	0
USER6	Mujer	novato	JS	10	10	10	0	10	600	fallo	600	fallo	600	fallo	600	fallo	600	fallo	600	fallo	0	6	0
			Media	8,00	8,14	8,00	1,57	8,86	600,00		558,33		600,00		470,00		380,00		370,00		0,50	5,50	7,17
			Desviacion estandar	1,33	1,60	0,98	1,17	0,55	0,00		102,06		0,00		209,28		259,23		269,44		1,22	1,22	8,59
			Error estandar	0,54	0,65	0,40	0,48	0,22	0,00		41,67		0,00		85,44		105,83		110,00		0,50	0,50	3,51
			Total Fallo							6		5		6		6		5		5			
			% Fallo							100		83		100		100		83		83			
			Total Éxito							0		1		0		0		1		1			
			% Éxito							0		17		0		0		17		17			

Ilustración 85- Resultados sin la herramienta

A continuación, se mostraran los resultados obtenidos en el caso de realizar la prueba utilizando la herramienta:

	HERRAMIENTA DE AYUDA																			
	NASA-TLX					TASK														
	1	2	3	4	5	Tiempo T1	Exito T1	Tiempo T2	Exito T2	Tiempo T3	Exito T3	Tiempo T4	Exito T4	Tiempo T5	Exito T5	Tiempo T6	Exito T6	Total Éxito	Total Fallo	Consultas documentacion
Participantes	7	3	5	7	2	120	exito	30	exito	60	fallo	30	fallo	20	exito	20	exito	4	2	0
USER1	3	3	1	9	2	90	fallo	30	exito	30	exito	30	exito	20	exito	40	exito	5	1	0
USER2	6	5	2	8	1	180	exito	60	exito	60	exito	30	exito	60	exito	30	exito	6	0	3
USER3	5	3	2	8	2	120	exito	60	exito	120	exito	60	exito	90	exito	90	exito	6	0	0
USER4	7	6	8	10	2	300	exito	60	exito	300	exito	300	exito	60	fallo	300	exito	5	1	0
USER5	4	4	4	10	3	120	exito	60	exito	120	exito	60	exito	60	exito	60	exito	6	0	0
USER6																				
Media	4,71	3,71	3,57	8,00	2,43	155,00		50,00		115,00		85,00		51,67		90,00		5,33	0,67	0,50
Desviacion estandar	1,63	1,26	2,58	1,21	0,63	76,88		15,49		97,52		106,35		27,14		105,83		0,82	0,82	1,22
Error estandar	0,67	0,52	1,05	0,49	0,26	31,38		6,32		39,81		43,42		11,08		43,20		0,33	0,33	0,50
Total Fallo							1		0		1		1		1		0			
% Fallo							17		0		17		17		17		0			
Total Éxito							5		6		5		5		5		6			
% Éxito							83		100		83		83		83		100			

Ilustración 86- Resultados con la herramienta