Trabajo de Fin de Grado

Sistemas y Tecnologías Web: Gitbook-plugin-Jazer

Jazer Abreu Chinea

La Laguna, 26 de mayo de 2016

D. **Casiano Rodríguez León,** con N.I.F. 42.020.072-S profesor Titular de Universidad adscrito al Departamento de Nombre del Departamento de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"Sistemas y Tecnologías Web: Gitbook-plugin-jazer"

ha sido realizada bajo su dirección por D. **Jazer Abreu Chinea,** con N.I.F. 78.853.499-T.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 4 de marzo de 2015.

Agradecimientos

Casiano Rodríguez león

Zulay Chinea Barroso

Fernando Abreu Piñero

Licencia

Resumen

Este proyecto trata sobre la creación de un plugin de Gitbook para la elaboración de ejercicios, con las validaciones basadas en expresiones regulares o funciones en lenguajes como JavaScript, Java, Ruby, Python y C++ con la posibilidad de extenderlo para soportar más lenguajes en un futuro y solucionando los problemas de seguridad que surjan.

Palabras clave: *Gitbook, plugin, ejercicios, expresiones regulares, JavaScript*

Abstract

This project is about the creation of a plugin for Gitbook, and allow the user to elaborate exercises with the validations based on regular expressions or JavaScript, Java, Ruby, Python and C++ functions, with the possibility of adding more languages in a future and solving the security problems we encounter.

Keywords: *Gitbook, plugin, exercises, regexp, JavaScript*

Índice General

Índice de figuras

Indice de tablas

Capítulo 1. Introducción

1.1 ¿Qué es Gitbook?

Gitbook es una herramienta que permite crear libros y documentos utilizando sintaxis Markdown o AsciiDoc. Estos libros una vez creados, pueden ser visualizados en web, o pueden ser exportados como e-book o pdf.

Está programada totalmente en NodeJS y está disponible actualmente bajo licencia Apache 3.0.

Actualmente también ofrece una aplicación de escritorio que ayuda a crear el libro y facilita su publicación.

Aparte, esta herramienta ofrece una especie de plataforma web que, tras identificarnos nos permite publicar los libros de forma gratuita para que cualquier persona pueda visualizarlo. Esta característica nos ofrece funciones adicionales como la posibilidad de escribir comentarios en los párrafos del libro a cualquier persona previamente registrada y que así lo desee.

Para publicar un libro simplemente hay que hacer un git push, tal y como lo haríamos para subir un proyecto a

Github. Por lo que ellos mismo definen su plataforma como un Github para libros.

Hay que aclarar que este servicio es opcional y nuestro libro se podría publicar en cualquier plataforma (gh-pages, heroku o cualquier servidor personal)

Por último, han dado la posibilidad de ampliar las funcionalidades que ofrece Gitbook mediante la utilización de plugins. Más adelante se darán más detalles de la utilización y la instalación de los mismos.

1.2 Situación actual de los plugins en Gitbook

Gitbook actualmente ofrece muchísimos plugins, y mucho de ellos los han elaborado los mismos creadores de Gitbook.

Entre ellos encontramos plugins que permiten cosas como:

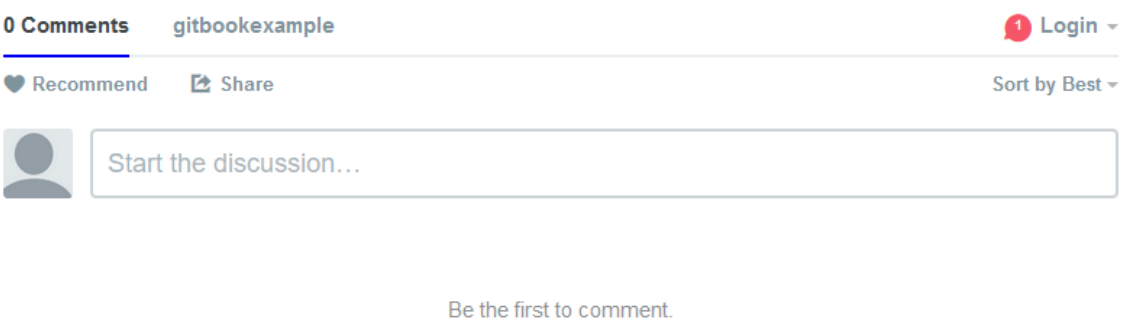
- Añadir sintaxis matemática mediante Katex.

```
Inline math: {% math %}\int_{-\infty}^\infty g(x) dx{% endmath %}
```

Inline math:

$$\int_{-\infty}^{\infty} g(x)dx$$

- Añadir comentarios mediante Disqus.



- Añadir un botón para copiar ciertos párrafos de texto.

```
s = "Ruby syntax highlighting"
    puts s
```

Copy

- Añadir videos de Youtube

Aparte hay plugins de ejercicios, pero sin la libertad de poder elaborar preguntas que se permitan validar de una forma compleja utilizando expresiones regulares o mediante funciones en cualquier lenguaje de programación.

1.3 Introducción a Gitbook

1.3.1 Instalación de Gitbook

Como hemos comentado anteriormente Gitbook es una herramienta de línea de comandos programada en NodeJS.

Para su utilización tenemos los siguientes requisitos:

- NodeJs v4.0.0 o superior
- Windows, Linux, Unix o Mac OS X

Para su utilización una vez tengamos instalado NodeJS y npm debemos abrir una terminal y escribir el siguiente comando:

```
npm install gitbook-cli -g
```

gitbook-cli es una utilidad que permite instalar y utilizar multiples versiones de Gitbook en un mismo sistema. Esta herramienta instalará la versión de Gitbook requerida para crear nuestro libro.

1.3.2 Crear un libro

Para crear un libro debemos situarnos con nuestra terminal dentro de la carpeta en la que queremos que se cree e introducir el siguiente comando:

```
gitbook init
```

Este comando creará los archivos necesarios para la utilización de nuestro libro. Entre ellos un `README.md` y un `SUMMARY.md`

En el fichero `README.md` se escribe el libro en formato Markdown y en el fichero `SUMMARY.md` se define la estructura del mismo. El libro puede tener varios ficheros readme para modularizar el libro y luego definir su orden en el fichero summary.

Archivo README

```
# Introduction

Introduccion del libro

bla bla bla
```

Archivo SUMMARY

```
# Summary

This is the summary of my book.

* [section 1](section1/README.md)
  * [example 1](section1/example1.md)
  * [example 2](section1/example2.md)
* [section 2](section2/README.md)
   * [example 1](section2/example1.md)
```

1.3.3 Construir el libro

Una vez tengamos nuestros archivos readme y summary bien definidos procedemos a construir el libro. Para ello tenemos varias opciones:

- Construirlo para obtener el libro en formato web

Para ello ejecutamos: **gitbook build**

Esto creará una carpeta con todos los archivos web.

- Construirlo para obtener una version en pdf

Para ello ejecutamos **gitbook pdf**

- Construirlo para obtener un libro en formato epub

Ejecutamos **gitbook epub**

- Construirlo para obtener un libro en formato mobi

Ejecutamos `gitbook mobi`

- Construirlo para obtener el libro en formato web y montarlo en un servidor provisional de forma local en el puerto 4000.

Ejecutamos `gitbook serve`

Para obtener una información más detallada acerca de la construcción del libro podemos utilizar la opción de debug:

```
gitbook build ./ --log=debug --debug
```

1.4 ¿Cómo funcionan los plugins de gitbook?

Un plugin de gitbook es simplemente un paquete de node que puede ser publicado en npm.

Para la instalación de un plugin debemos crear un archivo book.json, en este se indican los plugins que vamos a instalar como las configuraciones de los mismos.

Como su nombre indica en plugins se definen lo que queremos instalar y en pluginsConfig la configuración de cada uno.

Tras definir nuestro archivo debemos ejecutar el siguiente comando:

```
gitbook install
```

Gitbook internamente se encargará de descargar todos los plugins y dejarlo todo listo para que funcionen.

Capítulo 2. Objetivos y plan de trabajo

En este capítulo se detallarán los periodos estimados de trabajo y los objetivos presentes para este proyecto.

2.1 Objetivos

Los objetivos a desarrollar durante el proyecto se definen a continuación:

- **A1**. Estudiar el funcionamiento de Gitbook tanto como usuario como de forma más técnica.
- **A2**. Estudiar los plugins actuales en Gitbook y su funcionamiento.
- **A3**. Estudio de las tecnologías requeridas.
- **A4**. Estudiar cómo realizar nuestro propio plugin e incluir funcionalidades con él.
- **A5**. Creación de un plugin básico el cual permita añadir ejercicios a Gitbook. Todas las funcionalidades serán de prueba y sencillas en este punto.
- **A6**. Ampliar el plugin para que permita validarse mientras expresiones regulares.
- **A7**. Añadir la posibilidad de validar los ejercicios mediante una función JavaScript.

- **A8**. Ampliar la funcionalidad anterior para que permita más lenguajes como Python, Ruby, C++ y Java.

- **A9**. Implementar la seguridad del plugin. Más adelante veremos que para ello hemos necesitado crear un servidor.

Cada uno de estos objetivos tendrán unas fechas estipuladas para realizarse.

2.2 Plan de trabajo

A continuación, se indican las fechas previstas para cada objetivo.

Objetivo	Fecha
A1	5-10 Febrero
A2	10-20 Febrero
A3	20-29 Febrero
A4	1-10 Marzo
A5	10-31 Marzo
A6	1-15 Abril
A7	15-30 Abril
A8	1-15 Mayo
A9	15 Mayo-20 Junio

Capítulo 3. Desarrollo del plugin

En este capítulo se presentará todo el proceso de desarrollo del plugin, así como sus funcionalidades y utilización.

3.1 Estudio del funcionamiento de un Plugin para Gitbook

Como hemos especificado anteriormente, para la elaboración del plugin hemos tenido que investigar sobre el funcionamiento interno de Gitbook.

Para ello tenemos una página de documentación para los Gitbook developers.

Un plugin de Gitbook es simplemente un paquete de NodeJS publicado en NPM.

La estructura del plugin debe empezar por un fichero package.json en la que se define el nombre, la versión del plugin, la descripción y la versión de Gitbook que se tiene que utilizar para que funcione.

```
{
    "name": "gitbook-plugin-mytest",
    "version": "0.0.1",
    "description": "This is my first GitBook plugin",
    "engines": {
        "gitbook": ">1.x.x"
    }
}
```

El nombre del plugin tiene que empezar con gitbook-plugin-* y si es un tema para gitbook tiene que contener gitbook-theme-*.

Decir que este **package.json** es uno básico y en la página de NPM hay muchas más opciones e información acerca de su configuración.

El punto de entrada del plugin es el fichero index.js y tiene la siguiente estructura:

```
module.exports = {
    // Map of hooks
    hooks: {},

    // Map of new blocks
    blocks: {},

    // Map of new filters
    filters: {}
};
```

A continuación, explicaré cada uno de estos elementos.

3.1.1 Blocks

Un bloque simplemente es una etiqueta que posteriormente es procesada por una función. Esta función devolverá código HTML que sustituirá a esta etiqueta al construirse el libro.

Toda etiqueta debe tener una función process.

Imaginemos que tenemos un fichero **README.md** con la siguiente etiqueta

```
{% tag1 %}
Jazer
{% endtag1 %}
```

Todas las etiquetas siguen la misma estructura:

{% nombre %}
Contenido
{% endnombre %}

Y tenemos un plugin que en la parte de blocks se define lo siguiente

```
module.exports = {
    blocks: {
        tag1: {
            process: function(block) {
                return "Hello "+block.body+", How are you?";
            }
        }
    }
};
```

Vemos que se define un bloque llamado tag1 que es el nombre que debe tener la etiqueta en el fichero **README.md.**

Este bloque en la función **process** se le pasa un parámetro **block.** Ese parámetro es un objeto que contiene toda la información de la etiqueta que está procesando.

Esta función al final devuelve una línea en la que se interpola **block.body,** y contiene el texto que se ha definido en la etiqueta, en este caso **"Jazer".**

El resultado final sería **"Hello Jazer, How are you?"** al construir el libro.

Parámetros en el bloque

El bloque puede contener parámetros de la siguiente forma:

```
{% tag1 "argument 1", "argument 2", name="Test" %}
This is the body of the block.
{% endtag1 %}
```

Se procesaría en el index.js de la siguiente manera

```
module.exports = {
    blocks: {
        tag1: {
            process: function(block) {
                // block.args equals ["argument 1", "argument 2"]
                // block.kwargs equals { "name": "Test" }
            }
        }
    }
};
```

Dentro de la variable ***block.args*** se encontrarían los parámetros sin nombre como **"argument 1"** y en ***block.kwargs*** es un objeto que contiene los parámetros con nombre como ***"name='Test'"***.

Sub-bloques

Por ultimo comentar que cada bloque puede contener sub-bloques de la siguiente manera:

```
{% myTag %}
    Main body
    {% subblock1 %}
    Body of sub-block 1
    {% subblock 2 %}
    Body of sub-block 1
{% endmyTag %}
```

3.1.2 Filters

Los filters o filtros son esencialmente funciones que se pueden aplicar a variables.

Tienen la siguiente forma:

```
{{ "contenido" | nombredelfiltro(argumentos) }}
```

Teniendo un filter definido de la siguiente forma en nuestro index.js:

```
module.exports = {
    filters: {
        hello: function(name) {
            return 'Hello '+name;
        }
    }
};
```

Y teniendo un **README.md** que contiene lo siguiente:

```
{{ "Aaron"|hello }}, how are you?
```

Vemos que en nuestro **index.js** el filtro **hello** toma el parámetro, en este caso **“Aaron”** y lo sustituiría por la expresión **“Hello Aaron”.**

Parámetros

Los filtros pueden soportar parámetros de la siguiente forma:

```
Hello {{ "Samy"|fullName("Pesse", man=true}} }}
```

```
module.exports = {
    filters: {
        fullName: function(firstName, lastName, kwargs) {
            var name = firstName + ' ' + lastName;

            if (kwargs.man) name = "Mr" + name;
            else name = "Mrs" + name;

            return name;
        }
    }
};
```

“Samy” se sustituiría por **firstName.**

“Pesse” se sustituiría por **lastName.**

Y por último los parámetros con nombre están dentro del objeto **kwargs.**

Esto permite crear situaciones más complejas.

En este caso en el libro se vería reflejado lo siguiente:

“Mr Samy Pesse” si es hombre.

“Mrs Samy Pesse” si es una mujer.

3.1.3 Hooks

Los hooks son funciones que permiten alterar el contenido del libro.

Los hooks permiten alterar el contenido en varias fases de construcción del libro.

Hay 2 tipos, relativos al montaje global y al montaje de cada página.

Montaje global

Init

Este hook modifica el libro antes de que se genere el libro y las páginas.

Finish:before

Modifica el contenido después de generar el libro, pero antes de haber cambiado las etiquetas de bloques, filtros, incluir la portada y haber copiado el CSS propio del plugin.

Finish

Esta función permite cambiar el contenido del libro después de haberse generado todo lo demás.

Montaje de cada pagina

Page:before

Se permite modificar la página antes de que se procese cualquier cambio de filtro y bloque o CSS en ella.

Page

Se modifica la página después de haberse generado todo.

Ejemplos

Cambiar titulo de una página

```
{
    "page:before": function(page) {
        page.content = "# Title\n" +page.content;
        return page;
    }
}
```

Cambiar contenido de una página

```
{
    "page": function(page) {
        page.sections[0].content = page.sections[0].content.replace("<b>", "<strong>")
            .replace("</b>", "</strong>");
        return page;
    }
}
```

3.1.4 Recursos

Los recursos son archivos CSS o JavaScript que se añaden al construir el libro.

Se incluyen en el index.js de la siguiente manera:

```
module.exports = {
    book: {
        assets: './assets/website',
        css: [
            'mystyle.css'
        ],
        js: [
            'myfile.js
        ]
    }
};
```

El parámetro “book” indica que queremos que se incluyan cuando el libro se genera de forma web. Si quisiéramos que se incluya cuando se genera en pdf o ebook se debería llamar “ebook”.

En **assets** se indica la carpeta que contiene dichos archivos.

El parámetro css es un array que indica todos los archivos css que se van a incluir.

El parámetro js todos los archivos JavaScript.

3.2 Gitbook-plugin-jazer

En este apartado se explicará todas las funcionalidades del plugin que he desarrollado llamado “Jazer” para Gitbook.

3.2.1 Ejercicio validado con expresiones regulares

Este tipo de ejercicio, consiste en la posibilidad de que un autor defina una pregunta. Esta pregunta tras el usuario introducir la respuesta, se validará con una expresión regular definida por el autor.

A continuación, pondré un ejemplo básico de cómo se escribiría una pregunta de este estilo en el archivo README.md teniendo mi plugin instalado.

```
{% regexp %}
¿Who discovered America?
{% solution %}
Christopher Columbus
{% validation %}
/(\s*(Crist[oó]bal\s+)?Col[oó]n\s*)|((Christopher\s+)?Columbus)/i
{% editor %}
Placeholder text on editor
{% endregexp %}
```

Lo primero que encontramos es que se ha definido un bloque.

Este bloque tiene que llamarse **“regexp”** y acabar con **“endregexp”**

A continuación, tenemos la pregunta que el autor quiere que se le plantee al usuario. En este caso **“¿Who discovered America?”**

A partir de aquí tenemos sub-bloques.

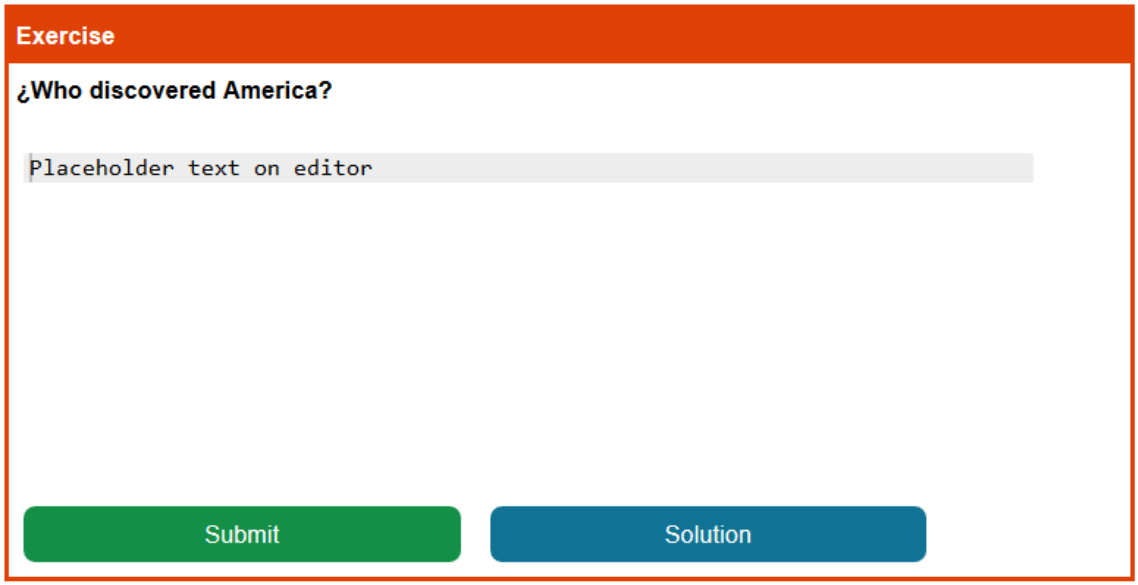
El primer sub-bloque es el **“solution”.** Este define la solución que se le muestra al alumno en el editor tras presionar el botón **“solution”** que veremos más adelante.

El sub-bloque **“validation”** indica la expresión regular que va a procesar el texto que ha introducido el usuario. Si

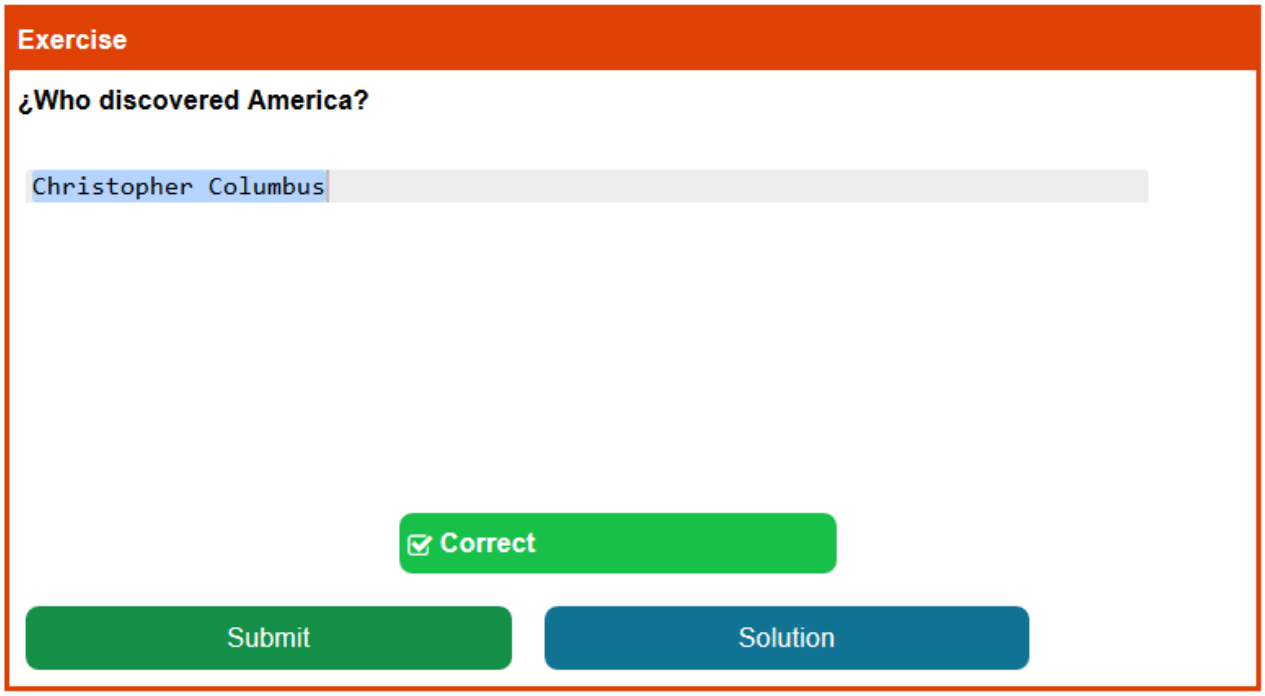
la expresión regular casa con el texto, la expresión devolverá verdadero y la respuesta será correcta.

El sub-bloque **“editor”** indica el texto que se va a mostrar en el editor al cargar el ejercicio. Lo que se llama en ingles el placeholder.

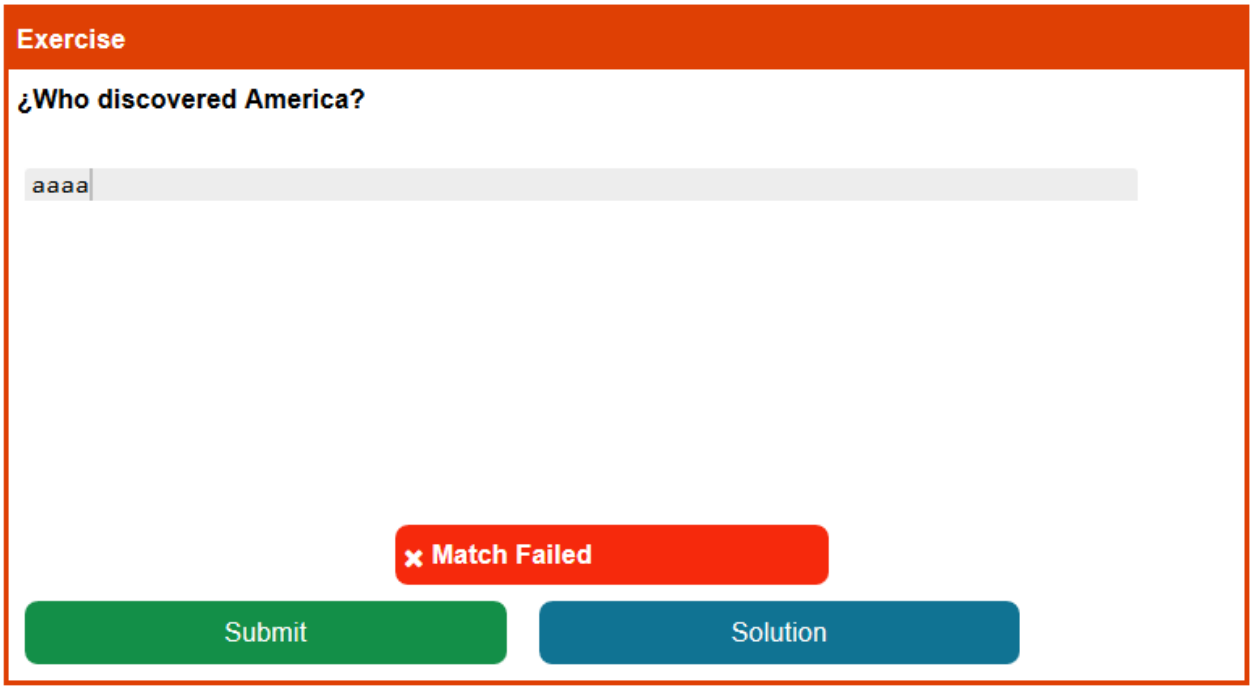
Tras crear el libro con el **README.md** anterior generaríamos lo siguiente:



Si le damos al botón **“solution”** y luego al botón **“submit”** veremos que la respuesta es correcta.



Si introducimos una respuesta incorrecta nos daría lo siguiente:



Este tipo de ejercicios soporta expresiones regulares complejas basadas en la librería de JavaScript XRegexp

A continuación, un ejemplo de un ejercicio utilizando este tipo de expresiones.

```
{% regexp %}
Who were the Spanish kings when America was discovered?
{% solution %}
Catholic Monarchs, also called Catholic Kings, or Catholic Majesties, Spanish Reyes Católicos, Ferdinand II of Aragon and Isabella I of Castile
{% validation %}
/
  (Catholic\s+Monarchs)           |
  (Catholic\s+Kings)              |
  (Catholic\s+Majesties)          |
  ((Spanish)?\s+Reyes\s+Católicos) |
  (Ferdinand(\s+II)?(\s+of\s+Aragon)?(\s+and)?(\s+Isabella)(\s+I)?(\s+of\s+Castill?e) |
  (Isabella)(\s+I)?(\s+of\s+Castill?e)(\s+and)?\s+(Ferdinand(\s+II)?(of\s+Aragon)?
/ix
{% editor %}
Placeholder text on editor
{% endregexp %}
```

3.2.2 Ejercicio validado con una función JavaScript

Este tipo de ejercicio surge de la necesidad de ejercicios que requieran de una validación con una complejidad superior a la que se puede manejar con una expresión regular.

A continuación, se muestra como se definiría este tipo de ejercicio.

```
{% questionjs %}
Who were the Spanish kings when America was discovered?
{% solution %}
Catholic Monarchs, also called Catholic Kings, or Catholic Majesties, Spanish Reyes Católicos, Ferdinand II of Aragon and Isabella I of Castile
{% validation %}
function(answer) {
  if (answer.match(/Catholic\s+(Monarchs|Kings|Majesties)/i)) return true;
  if (answer.match(/(Spanish\s+)?Reyes\s+Cat[oó]licos/i)) return true;
  if (answer.match(/isabel|isabella/i && respuesta.match(/fernando|ferdinand/i) )) return true;
}
{% editor %}
Placeholder text on editor
{% endquestionjs %}
```

Como podemos ver, lo único que cambia es el nombre del bloque principal, en este caso se pasaría a llamar **"questionjs".**

Los sub-bloques funcionan de la misma forma que funcionan para los ejercicios con expresiones regulares con excepción de **"validation".**

A este sub-bloque se le tiene que pasar una función JavaScript con un parámetro. Este parámetro contendrá el texto que introduzca el usuario. Tras definir como procesar este texto la función tiene que devolver **"true"** cuando la respuesta sea correcta y **"false"** o no devolver nada en cualquier otro caso.

La visualización del ejercicio es exactamente igual que para un ejercicio basado en expresiones regulares.

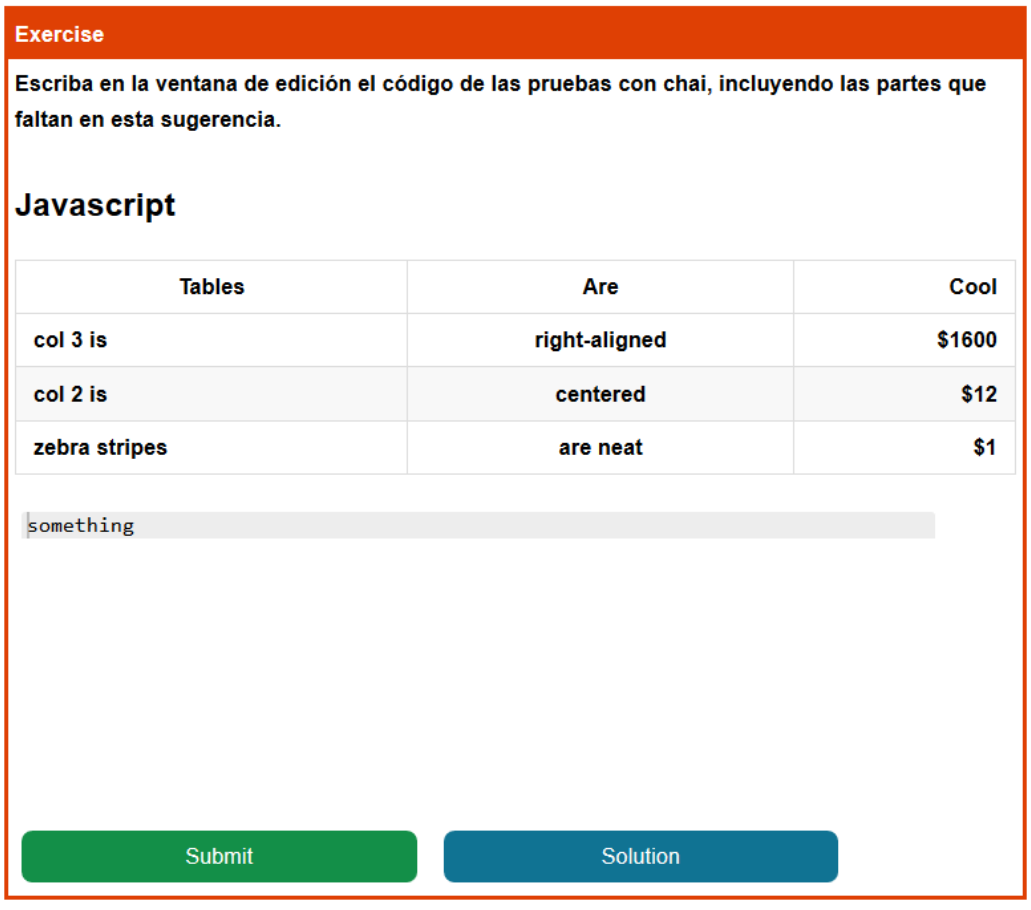
3.2.3 Los ejercicios soportan Markdown.

Cualquiera de los ejercicios que genera el plugin, su enunciado se puede definir con Markdown incluso con las funcionalidades de Github.

A continuación, se muestra un ejemplo.

```
{% regexp %}
Escriba en la ventana de  edición el código de las pruebas con chai,
incluyendo las partes que faltan en esta sugerencia.

### Javascript

| Tables        | Are           | Cool  |
| ------------- |:-------------:| -----:|
| col 3 is      | right-aligned | $1600 |
| col 2 is      | centered      |   $12 |
| zebra stripes | are neat      |    $1 |

{% editor %}
something
{% solution %}
something
{% validation %}
/
something
/ix
{% endregexp %}
```

Esto generaría un ejercicio que contiene un título en etiqueta html “h3” y aparte una tabla.



3.2.4 Parámetros

Se pueden definir una serie de parámetros que modifican tanto el aspecto visual como la funcionalidad de cada uno de los ejercicios.

Los parámetros se pueden definir para cada ejercicio individualmente o para todo el libro de forma global.

Los parámetros de cada ejercicio locales tienen prioridad sobre los parámetros definidos globalmente.

Hay seis parámetros disponibles: **"width", "color", "gutter", "editorHeight", "editorAutoHeight" y "fontSize".** De forma global hay un parámetro adicional llamado **"support".**

- El parámetro **"width"** debe ser un porcentaje y modifica el ancho que ocupa el ejercicio.
- El parámetro **"color"** modifica el color que tiene el marco del ejercicio y puede ser cualquier valor css valido.
- El parámetro **"gutter"** debe ser "true" o "false" y decide si mostrar o no los números de líneas en el editor.
- El parámetro **"editorHeight"** debe ser un tamaño en pixels ("200px") y modifica la altura del editor.
- El parámetro **"editorAutoHeight"** puede tener el valor **"solution"** o **"editor"** y hace que el plugin calcule automáticamente la altura del editor basado en el tamaño del contenido del sub-bloque **"solution"** o del sub-bloque **"editor"** respectivamente.
- Por último, el parámetro **"fontSize"** puede tener valores como "16px" o "16" y especifica el tamaño de la letra en pixeles.
- Parámetro **"support"** solo disponible de forma global, permite definir un array de cadenas, las cuales contienen librerías de soporte para utilizar en los

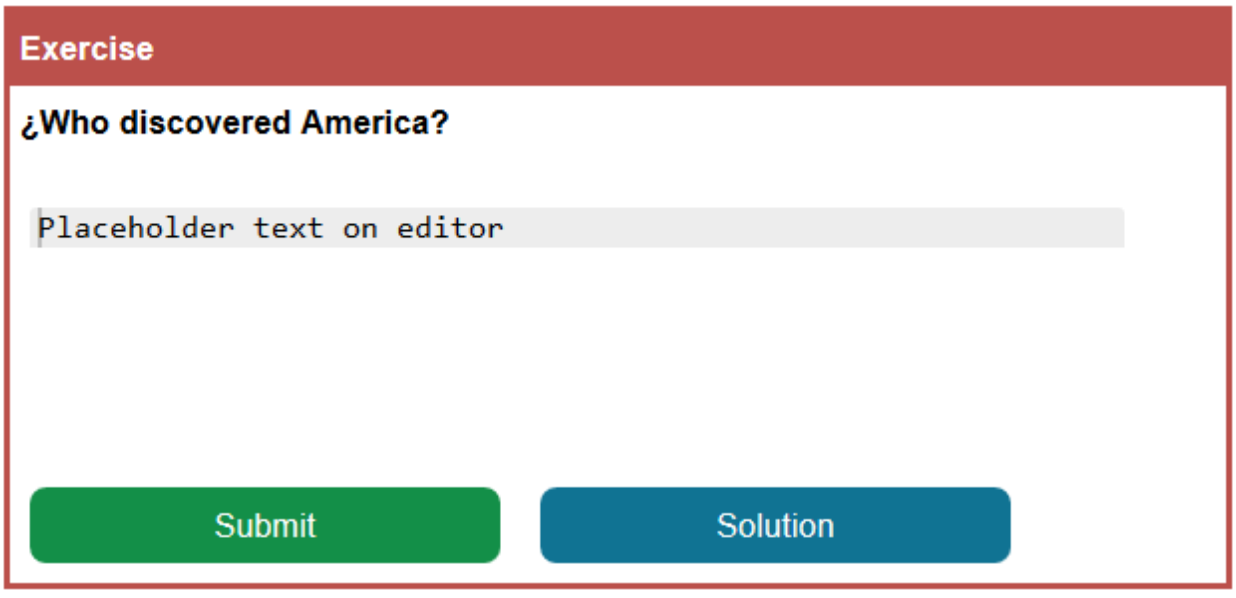
ejercicios de funciones JavaScript. Estos recursos pueden ser por ejemplo direcciones cdn's a dichas librerías.

Para definir los parámetros globales se tiene que modificar el archivo **book.json** e incluir los parámetros de la siguiente forma:

```
{

  "plugins": ["jazer"],

  "pluginsConfig":{
      "jazer": {
          "width": "80%",
          "color": "#BB504B",
          "gutter": "false"
          "support": ["https://ajax.googleapis.com/ajax/libs/angularjs/1.4.9/angular.min.js","https://ajax.googleapis.com/ajax/libs/mootools/1.6.0/mootools.min.js"],
          "editorHeight": "100px" //or editorAutoHeight: "solution",
          "fontSize": "16px"


      }
  }

}
```

En el parámetro **"pluginsConfig"** y luego dentro de **"jazer"** vemos como están definidos todos los parámetros anteriormente mencionados.

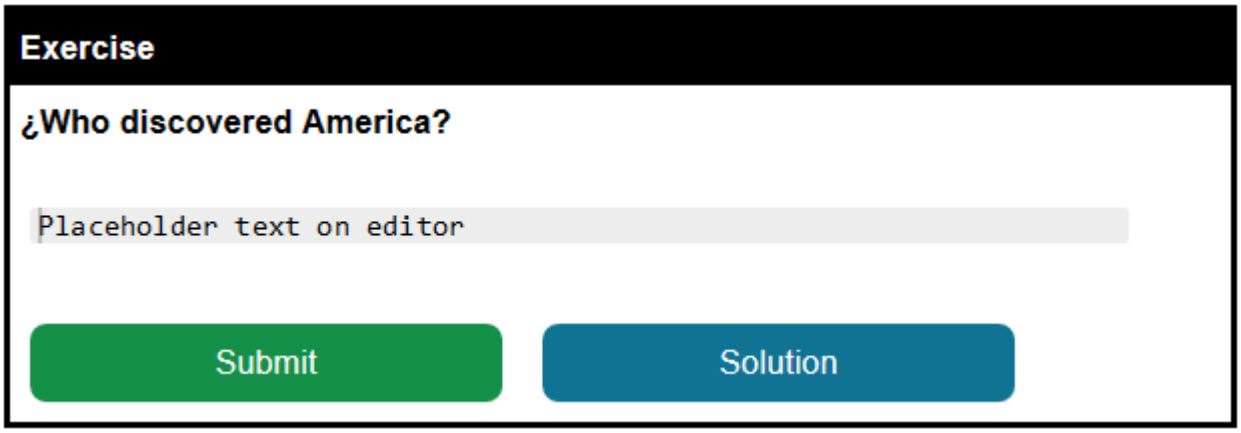
A continuación, vemos como se vería un ejercicio con estos parámetros definidos.



Todos estos parámetros o algunos pueden ser definidos de forma local a cada ejercicio de la siguiente forma:

```
{% regexp color="black", fontSize="15px", editorAutoHeight="solution" %}
¿Who discovered America?
{% solution %}
Christopher Columbus
{% validation %}
/(\s*(Crist[oó]bal\s+)?Col[oó]n\s*)|((Christopher\s+)?Columbus)/i
{% editor %}
Placeholder text on editor
{% endregexp %}
```

Con el siguiente resultado:



Como vemos, los parámetros de **“color”**, **“fontSize”** y **“editorAutoHeight”** locales, sobrescriben a los estipulados en el fichero **book.json.**

Por ultimo comentar que si **“editorHeight”** es especificado, entonces este tiene prioridad sobre **“editorAutoHeight”**, pero **“editorAutoHeight”** especificado localmente en el bloque tiene prioridad sobre **“editorHeight”** especificado en el **book.json.**

3.2.5 Ejercicios validados mediante funciones en cualquier lenguaje.

Estos ejercicios se definen de la siguiente forma en el README.md

```
{% question  width="60%", color="#BB504B"%}
¿Quienes reinaban en España cuando se descubrió America? RUBY
{% solution %}
reyes catolicos
{% validation %}
def exercise(respuesta)
  if(respuesta =~ /something/)
    return true;
  else
    return false;
  end
end
{% language %}
ruby
{% endquestion %}
```

Como podemos ver la diferencia está en el nombre del bloque principal que pasa a llamarse **“question”.**

En el sub-bloque **“validation”** se define la función en el lenguaje que queramos. Esta función debe tener el nombre **“exercise”,** devolver un **“true”** o **“false”** y tener un parámetro en el que se le pase una cadena de texto.

Por ultimo tiene un sub-bloque adicional llamado **“language”** en el que se especifica el lenguaje en el que queremos validar la pregunta.

Por ahora se soporta ruby, javascript, c++ , python y java.

La visualización de este ejercicio es exactamente igual a los anteriores.

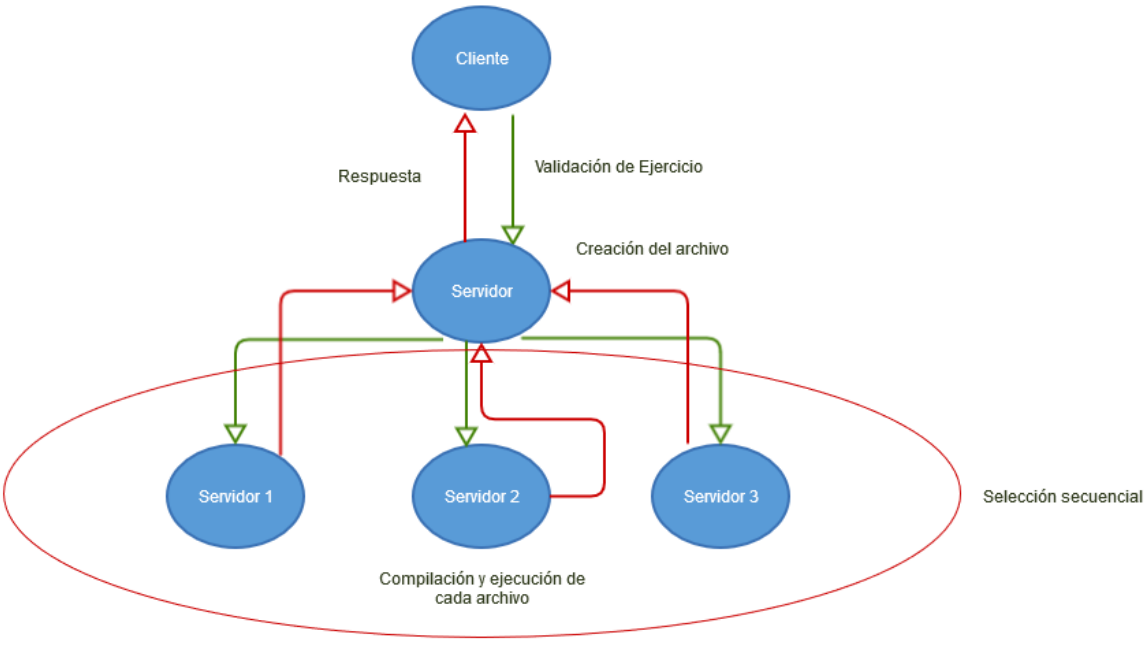
Comentar que para que esto funcione hace falta de un servidor externo al que se conecta el plugin que se procederá a explicar en el siguiente punto.

3.2.6 Servidor de validación del bloque question

Para validar todos estos tipos de lenguaje se tiene un servidor externo el cual está programado en NodeJS. Se basa en la infraestructura web Express que facilita la creación de este tipo de servidores.

Comentar que aparte se necesita de una infraestructura de servidores adicionales para evitar cualquier tipo de código malicioso y asegurar así nuestro servidor.

A continuación, se presenta un esquema del funcionamiento del mismo:



Como vemos el cliente cuando quiere validar una pregunta, hace una petición al servidor.

El servidor obtiene la validación, la respuesta del usuario y el lenguaje en el que se quiere realizar dicha validación.

Este crea el archivo en el lenguaje que se le ha indicado y lo sube a un servidor de apoyo, estos servidores se eligen de

forma secuencial para cada una de las preguntas que se vayan a validar.

Este servidor al recibir el fichero, lo ejecuta, le devuelve la respuesta al servidor principal y tras acabar todo el proceso borra todos los archivos generados, tanto en el principal como en el servidor de apoyo.

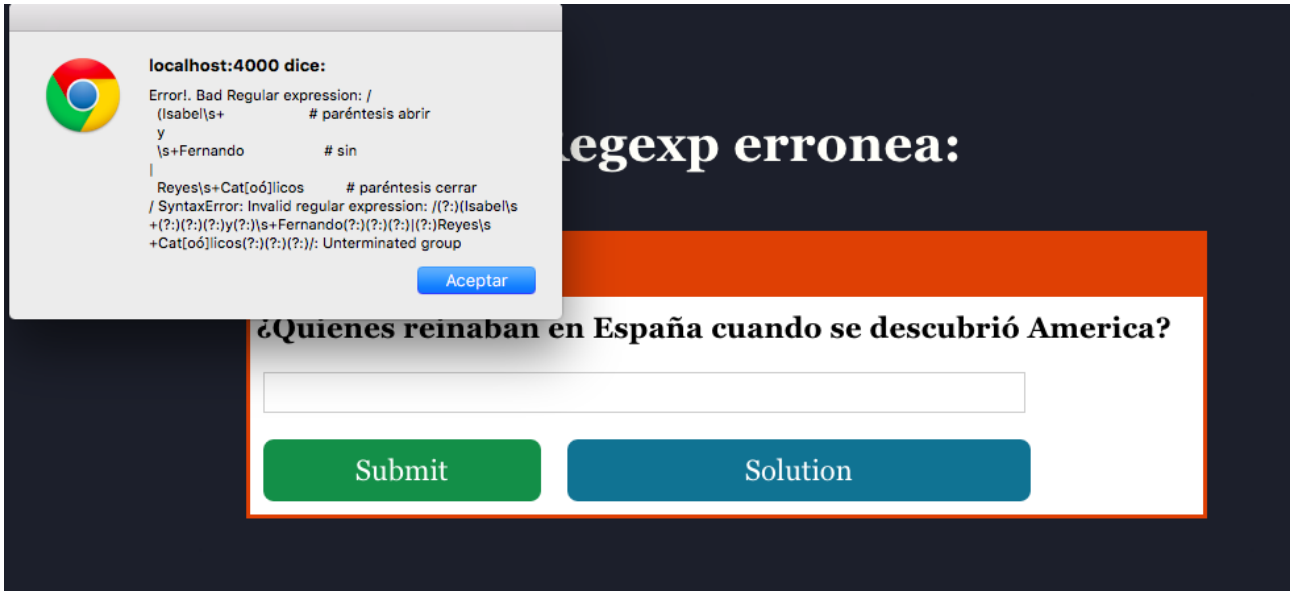
El servidor principal devuelve la respuesta al cliente, que en todos los casos será “true” o “false” y se le indicará el resultado al usuario.

Decir también que si surge cualquier tipo de error en todo el proceso se le mostrará al usuario cual ha sido este error en forma de alerta.

3.2.7 Gestión de errores para todos los ejercicios

Para todos los ejercicios, al usuario se le mostrará un error si en cualquiera de ellos ha habido algún problema de validación.

A continuación, una imagen:



3.3 Problemas encontrados y soluciones

3.3.1 Ejercicios con expresiones regulares de forma extendida

Se requería que los ejercicios permitieran expresiones regulares basadas en XRegExp.

Entre otras cosas XRegExp permite añadir las siguientes opciones a la expresión regular:

- s, hace que el (.) case con cualquier símbolo;
- x, permite añadir espacios y comentarios;
- n, modo de captura explicita
- A, modo astral (soporta 21-bit Unicode matching).

Para más información se puede consultar el siguiente enlace: http://xregexp.com/

La solución que se propuso fue añadir la librería a los diferente ficheros JavaScript que carga el plugin de forma estándar.

Por lo que puede ser usado por cualquier usuario.

3.3.2 Forma de mostrar los errores

Todos los ejercicios necesitaban una forma de mostrar los posibles errores al encontrarse con una expresión regular mal formada o una función en cualquier lenguaje que tenga algún error.

Para ello la opción más fácil y que mejor se adaptaba a la situación fue añadir una alerta que se le muestre al usuario al cargar la página web con el plugin.

En esta alerta se mostraría el error que se ha producido y en qué ejercicio se produjo.

Para más información se puede ir al punto 3.2.7 de este mismo documento.

3.3.3 Visualización del ejercicio al construirse un pdf o ebook

Se necesitaba de alguna forma poder mostrar el ejercicio cuando se genera el pdf o el ebook.

Para ello se añadió un template completamente diferente el cual contiene el ejercicio, pero con la respuesta ya incluida en el y sin ningún tipo de interacción.

Esto era necesario ya que pdf y ebook no tienen ningún tipo de lenguaje de programación que permita añadir funcionalidades.

3.3.4 Utilizar editor ACE como forma de introducir respuestas

En las versiones más tempranas del plugin se utilizaba un simple input para introducir las respuestas.

Más adelante se mejoró este aspecto utilizando un editor llamado ACE.

Permitía tener temas específicos, número de líneas y muchas más funciones.

Para más información se puede consultar el siguiente enlace: https://ace.c9.io/

3.3.5 Personalización de los aspectos visuales del plugin

En las primeras versiones todos los ejercicios tenían exactamente el mismo aspecto y no ofrecía ningún tipo de personalización.

Esto se mejoró más adelante al permitir introducir en cada ejercicio y de forma global parámetros al usuario que permitiesen cambiar el color, el ancho, altura, tamaño de fuente, entre otras cosas.

Para más información se puede ir al punto 3.2.4 de este documento.

Capítulo 4. Conclusiones y líneas futuras.

A pesar de existir otros plugins que permiten realizar ejercicios, mi trabajo ha permitido ampliar y dar más funcionalidades a este campo en Gitbook.

Decir que esto me ha permitido tener más experiencia tanto en NodeJS como en toda la herramienta de Gitbook, permitiéndome tener más conocimientos tanto técnicos como a nivel de usuario. También decir que al tener que suplir las necesidades de seguridad, he obtenido mucha experiencia en la creación de servidores y la comunicación entre los mismos.

Tengo que comentar que Gitbook es una herramienta muy buena en cuanto a creación de libros se refiere y creo que debería utilizarse más en todos los ámbitos educativos.

En mi trabajo futuro sobre el plugin, tengo pensado añadir más opciones y añadir la posibilidad de ejecutar más lenguajes de los que ya han sido implementados.

Agradecer toda la ayuda prestada por mi tutor Casiano Rodríguez León, que me ha resuelto muchas dudas y me ha dado muchísimas ideas para añadir y bugs que resolver durante toda la etapa de desarrollo.

Por ultimo comentar que el plugin está siendo utilizado ampliamente por mucha gente, ya que actualmente tiene 51 descargas diarias, 291 descargas semanales y 3572 descargas mensuales.

Gracias a todos por leer esta memoria y espero que les haya gustado mi trabajo.

Capítulo 5.
Summary and Conclusions

Although there are other plugins that allow exercises, my work has expanded and give more functionality to this field on Gitbook.

This let me to have more experience in NodeJS and throughout Gitbook tool, allowing me to have more technical and user-level knowledge. Also say for solving security needs, I have gained a lot of experience in creating servers and communication between them.

I have to comment that Gitbook is very good in terms of book creation tool and I think it should be used more in all educational areas.

In my future work on the plugin, I have plans to add more options and add the ability to run more languages which have already been implemented.

Thanks to all the help given by my tutor Casiano Rodriguez Leon, who has solved many doubts and given me a lots of ideas for adding and bugs to solve throughout the development stage.

Finally comment that the plugin is being widely used by many people, as it currently has 51 daily downloads, 291 weekly downloads and 3572 monthly downloads.

Thank you all for reading this report and I hope you liked my work.

Capítulo 6. Presupuesto

En este capítulo se especificará todo el presupuesto del que se ha dispuesto.

6.1.1 Coste en función del tiempo empleado

Se han empleado unos 4 meses de trabajo para desarrollar todo el proyecto. Esto supone de unos 1600 euros mensuales de un sueldo medio si fuesen remunerados.

Teniendo un total de unos 6400 euros de coste en el desarrollador.

6.1.2 Servidores externos

El STIC de la ULL, nos ha proporcionado 5 servidores los cuales he utilizado para realizar toda la parte de seguridad y ejecución de código de otro tipo de lenguajes como Java, C++, Python y Ruby.

Estos servidores se basan en máquinas Linux y están basados en máquinas virtuales OVirt.

Si hubiese que contratar un servicio externo de 5 servidores en una compañía como Heroku nos saldría el servicio más básico unos 300 euros mensuales.

Teniendo un total de 1200 euros por la duración de todo el desarrollo del proyecto.

6.1.3 Coste total

Teniendo en cuenta el coste en función de la remuneración del desarrollador y el alquiler mensual de los 5 servidores, tendríamos un coste total por el desarrollo de este proyecto de unos 7600 euros.

6.1.4 Obtención de ingresos por el plugin

Después de tener toda la herramienta terminada hay varias posibilidades de obtener algún tipo de ingreso por el uso de la misma.

La primera es la posibilidad de que los creadores de Gitbook tomaran mi servidor de forma que fuese estándar a su plataforma. De forma que todo el que quiera utilizar mi plugin y ejecutar cualquier tipo de lenguaje lo pudiese hacer a cambio de algún tipo de remuneración.

La segunda es ofrecer algún tipo de ventaja a aquellos usuarios que así lo deseen. Estos privilegios podrían ser la ejecución de cualquier número de peticiones paralelas o que hubiese algún límite con el numero diario de ejercicios que se puedan ejecutar en mi servidor.

En general estas opciones se tendrán en cuenta más adelante.

Apéndice A.

A.1. Repositorio del plugin

https://github.com/sokartema/gitbook-plugin-jazer

A.2. Repositorio del servidor

https://github.com/sokartema/plugin-server

A.3. Página de NPM

https://www.npmjs.com/package/gitbook-plugin-jazer

Bibliografía

[1] Gitbook plugin developers.
https://developer.gitbook.com/plugins/index.html
[2] Ace Editor https://ace.c9.io
[3] XRegExp http://xregexp.com/
[4] Lodash https://lodash.com/
[5] GruntJS http://gruntjs.com/
[6] ExpressJS http://expressjs.com/es
[7] StackOverflow http://stackoverflow.com/
[8] NodeJs https://nodejs.org/en/