



Sistemas y Tecnologías Web (SyTW)

Aplicación para la elaboración y despliegue de cuestionarios.

Application for the generation and deployment of questionnaires.

Juan José Labrador González

Ingeniería Informática y de Sistemas

Escuela Superior de Ingeniería y Tecnología

Trabajo de Fin de Grado

La Laguna, 6 de julio de 2014

D. **Casiano Rodríguez León**, con N.I.F. 42.020.072-S profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna

C E R T I F I C A

Que la presente memoria titulada:

“Sistemas y Tecnologías Web. Aplicación para la elaboración y despliegue de cuestionarios.”

ha sido realizada bajo su dirección por D. **Juan José Labrador González**, con N.I.F. 78.729.778-L.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 6 de julio de 2014

Agradecimientos

La realización de esta asignatura de Trabajo de Fin de Grado no hubiera sido posible sin la ayuda de la Sección de Ingeniería Informática de la Escuela Superior de Ingeniería y Tecnología que ha llevado a cabo todos los trámites necesarios.

Por otra parte, agradecer a Gara Miranda Valladares su labor, como Coordinadora de la asignatura de Trabajo de Fin de Grado, en el asesoramiento a los tutores y alumnos sobre los trámites y documentos a realizar, así como las fechas límite para sus entregas.

Mención especial para mi familia y amigos, que han caminado junto a mí durante todo este tiempo y me han alentado para no rendirme y lograr mis objetivos.

Y por último, especialmente agradecer a Casiano Rodríguez León su labor como tutor del Trabajo de Fin de Grado. Junto a él he aprendido nuevas tecnologías, conceptos y procedimientos a la hora de implementar aplicaciones. Me ha aconsejado, animado, motivado y resuelto mis dudas de manera incansable en la realización del Trabajo de Fin de Grado. Ha sido placer gozar de su conocimiento y experiencia. Estoy seguro de que la experiencia adquirida me ayudará en mis próximas etapas profesionales.

Resumen

El objetivo de este trabajo ha sido integrar los conocimientos adquiridos durante los estudios del Grado y, en especial, del itinerario de Tecnologías de la Información, aproximando al alumno a la resolución de problemas de aplicaciones Web y favoreciendo el desarrollo de destrezas propias de la Ingeniería Web: se centra en el aprendizaje y puesta en práctica de metodologías, aproximaciones, técnicas y herramientas para abordar la creciente complejidad de este tipo de aplicaciones en el marco de las metodologías ágiles.

En este Trabajo de Fin de Grado se propone el desarrollo de una gema de Ruby que facilite la elaboración y despliegue de cuestionarios autoevaluables. Éstos, además de poseer las típicas preguntas tipo test de respuesta única y multirespuesta, cuentan con la posibilidad de añadir preguntas de programación. Los resultados de la evaluación de los cuestionarios serán visibles para los profesores.

Para su desarrollo se ha partido de un Lenguaje de Dominio Específico (DSL) implementado en Ruby por Armando Fox denominado 'Ruby-based Quiz Generator and DSL' (RuQL).

Palabras clave: Generación de cuestionarios, Lenguaje de Dominio Específico, DSL, Ruby, RuQL, Sinatra.

Abstract

Here should be the abstract in a foreing language...

Keywords: *Keyword1, Keyword2, Keyword2, ...*

Índice general

1. Introducción	1
1.1. Antecedentes y estado actual del tema	1
1.2. ¿Qué es RuQL?	2
1.2.1. Código de ejemplo para realizar un cuestionario HTML5 . . .	4
1.3. Objetivos y actividades a realizar	6
1.4. Tecnología usada	7
2. Desarrollo	9
2.1. Metodología usada	9
2.1.1. GitHub	9
2.1.2. Testing	12
2.1.3. Experiencia de usuario	12
2.2. Resultados	13
3. Mejoras del DSL original	15
3.1. Problemas encontrados y soluciones	16
3.1.1. Entender el funcionamiento del código de la gema	16
3.1.2. Corregir tests y funcionalidades de la gema	16
4. HtmlForm renderer	17
4.1. Problemas encontrados y soluciones	27
4.1.1. Corrección de preguntas de Ruby en JavaScript	27
4.1.2. Alojamiento librería MathJax en un directorio de la gema	27
5. Sinatra renderer	29
5.1. Problemas encontrados y soluciones	31
5.1.1. Timeout corto entre peticiones del navegador al servidor . . .	31
5.1.2. Lugar de almacenamiento de las respuestas de los alumnos . .	32
5.1.3. Problema de seguridad al evaluar código Ruby en el servidor .	32
6. Conclusiones y trabajos futuros	33

7. Summary and Conclusions	35
7.1. First Section	35
8. Presupuesto	37
8.1. Sección Uno	37
A. Glosario de términos	39
A.1. T	39
B. Guía de usuario final	41
B.1. Guía de usuario: Seccion 1	41
Bibliografía	41

Índice de figuras

1.1. Pregunta de completar simple	3
1.2. Pregunta de completar con distractor y explicación	3
1.3. Pregunta multirrespuesta con una única respuesta correcta	3
1.4. Pregunta multirrespuesta usando la opción <i>raw</i>	4
1.5. Pregunta multirrespuesta con una múltiples respuestas correctas	4
1.6. Pregunta de verdadero o falso	4
2.1. Captura del repositorio de la gema en GitHub	10
2.2. Ramas del repositorio propio	10
2.3. Pull Request aceptado y cerrado	11
2.4. Contribuciones hechas al repositorio original	11
2.5. Apartado de issues cerrados	12
4.1. Código para incluir un header personalizado	18
4.2. Código para incluir un footer personalizado	18
4.3. Ejemplo de pregunta con HTML escapado	18
4.4. Ejemplo de pregunta con código LaTeX	19
4.5. Ejemplo de pregunta con código LaTeX renderizada	19
4.6. Ejemplo de pregunta con respuesta numérica	20
4.7. Ejemplo de pregunta con respuesta de tipo JavaScript	20
4.8. Ejemplo de pregunta con respuestas de múltiples longitudes	20
4.9. Ejemplo de pregunta con respuestas de múltiples longitudes renderizada	21
4.10. Ejemplo del primer tipo de pregunta simplificada	21
4.11. Ejemplo del segundo tipo de pregunta simplificada	21
4.12. Ejemplo de pregunta de completar con Drag and Drop	22
4.13. Ejemplo de pregunta de completar con Drag and Drop renderizada	22
4.14. Ejemplo de pregunta tipo test de respuesta única con Drag and Drop	23
4.15. Ejemplo de pregunta tipo test de respuesta única con Drag and Drop renderizada	23
4.16. Ejemplo de pregunta de tipo test multirrespuesta con Drag and Drop	24
4.17. Ejemplo de pregunta de tipo test multirrespuesta con Drag and Drop renderizada	24
4.18. Ejemplo de pregunta de programación	24
4.19. Ejemplo de pregunta de programación renderizada	25

4.20. Ejemplo de pregunta corregida	25
4.21. Puntuación total	26
4.22. Ejemplo de mostrar respuesta correcta en pregunta de completar . . .	26
4.23. Ejemplo de mostrar respuesta correcta en pregunta tipo test	26
4.24. Mensaje en español de Local Storage borrado	27
5.1. Método para especificar los profesores permitidos en la aplicación . .	29
5.2. Método para especificar los alumnos permitidos en la aplicación . . .	29
5.3. Método para especificar la configuración de la aplicación	30
5.4. Listado de ficheros y directorios generado dentro del directorio <i>app</i> .	30

Índice de tablas

8.1. Tabla resumen de los Tipos	37
---	----

Capítulo 1

Introducción

1.1. Antecedentes y estado actual del tema

La World Wide Web esta sujeta a un cambio continuo. La llegada de HTML5, la creciente importancia de AJAX y de la programación en el lado del cliente, las nuevas fronteras de la Web Semántica, la explosión de las redes sociales así como la llegada de las redes sociales federadas son ejemplos de esta tendencia general. Las aplicaciones web parecen evolucionar hacia entornos cada vez más ricos y flexibles en los que los usuarios pueden acceder con facilidad a los documentos, publicar contenido, escuchar música, ver vídeos, realizar dibujos e incluso jugar usando un navegador. Esta nueva clase de software ubicuo no cesa de ganar *momentum* y promueve nuevas formas de interacción y cooperación.

Dentro del ámbito educativo también se ha vivido una revolución, en forma y contenido, de impartir enseñanza. En Internet se puede encontrar, cada vez con más facilidad, contenidos de cualquier disciplina en múltiples formatos: blogs, videotutoriales, presentaciones, ejercicios resueltos, etc.

También están teniendo mucho éxito las plataformas de aprendizaje virtual ofreciendo, además de conocimiento sin la necesidad de estar físicamente presente en un aula, una serie de recompensas y medallas por ir obteniendo logros. Esta metodología se denomina **Gamificación** y está teniendo un impacto muy positivo en los usuarios de estas plataformas, ya que los anima a seguir usando estas herramientas de conocimiento.

Otro tipo de plataforma de aprendizaje virtual más orientada a universidades e institutos es **Moodle**. Está implantada en numerosos centros de todo el planeta. Es el complemento perfecto para enriquecer las enseñanzas impartidas físicamente con cuestionarios autoevaluativos y compartición de recursos adicionales. Además, facilita numerosas tareas a los docentes como la corrección y calificación de ejercicios.

Sin embargo, esta plataforma sólo se limita a la evaluación de cuestiones triviales. Para la corrección de preguntas propias de las ramas de Ingeniería, como pueden ser la implementación de código, es necesaria la figura del profesor para evaluar dichas tareas.

Otro problema que presenta es su difícil portabilidad. Estamos hablando de un tipo de plataforma que sigue un esquema **cliente-servidor**, que no es fácilmente migrable a otras máquinas.

Estas desventajas se ven resueltas, haciendo uso de **RuQL**, en la **Aplicación para la Elaboración y Despliegue de Cuestionarios** que se presentará en esta memoria. Dicha aplicación está destinada a profesores con ciertos conocimientos en el ámbito de la programación y de la informática, aunque la curva de aprendizaje no es excesiva para el resto del profesorado.

1.2. ¿Qué es RuQL?

Esta aplicación de generación y despliegue de cuestionarios hace uso de una gema de Ruby creada por Armando Fox denominada 'Ruby-based Quiz Generator and DSL' (RuQL).

Inicialmente, esta gema permitía generar un cuestionario partiendo de un fichero **Ruby**, donde se redactaban las preguntas y respuestas haciendo uso de un **DSL**.

Poseía una serie de *renderers* que permitían generar los cuestionario en los siguientes formatos:

- Open EdX: formato *open source* listo para importar en plataformas de aprendizaje online como **EdX**.
- Versión HTML 5 imprimible: lista para ser impresa y rellenada por los usuarios. Se le podía pasar como argumento el path de una hoja de estilo para incorporarla al HTML de salida. Del mismo modo, se podía especificar el path de un template predefinido por el profesor de modo que las preguntas se renderizaran en el mismo.
- AutoQCM: formato listo para importar a **AMC** (*Auto Multiple Choice*), software libre que permite elaborar cuestionarios multirrespuesta.

Los tipos de preguntas que se podían especificar eran:

- **Preguntas de completar:** en las cuales los usuarios deben rellenar los espacios en blanco. Admitía respuestas de tipo *string* o *regexp*. Si existían múltiples espacios para rellenar, se especificaban las respuestas en forma de *array*, indicando además si el orden de las mismas influía. Permitía además especificar

respuestas falsas (*distractors*) con una explicación de la misma, de modo que si el alumno escribía dicho *distractor*, le apareciera la explicación de por qué esa respuesta era incorrecta.

```
fill_in :points => 2 do
  text 'The capital of California is ---'
  answer 'sacramento'
end
```

Figura 1.1: Pregunta de completar simple

```
fill_in do
  text 'The visionary founder of Apple is ---'
  answer /^ste(ve|phen)\s+jobs$/
  distractor /^steve\s+wozniak/, :explanation => 'Almost, but not quite.'
end
```

Figura 1.2: Pregunta de completar con distractor y explicación

- **Preguntas multirrespuesta con una única respuesta correcta:** las clásicas preguntas tipo test. Se podía aleatorizar el orden de las respuestas definido en el fichero de preguntas y asignarles explicaciones a los *distractors* de manera individual o asignar una explicación general para todos los *distractors*.

```
choice_answer :randomize => true do
  text "What is the largest US state?"
  explanation "Not big enough." # for distractors without their own explanation
  answer 'Alaska'
  distractor 'Hawaii'
  distractor 'Texas', :explanation => "That's pretty big, but think colder."
end
```

Figura 1.3: Pregunta multirrespuesta con una única respuesta correcta

Especificando además la opción *raw* a la pregunta, permitía incrustar dicho texto entre etiquetas `<pre>` HTML.

```
choice_answer :raw => true do
  text %Q{What does the following code do:


```

 puts "Hello world!"

```


}
  distractor 'Throws an exception', :explanation => "Don't be an idiot."
  answer 'Prints a friendly message'
end
```

Figura 1.4: Pregunta multirrespuesta usando la opción *raw*

- **Preguntas multirrespuesta con una múltiples respuestas correctas:** iguales a las preguntas multirrespuesta de opción única con la diferencia de que existe más de una respuesta correcta.

```
select_multiple do
  text "Which are American political parties?"
  answer "Democrats"
  answer "Republicans"
  answer "Greens", :explanation => "Yes, they're a party!"
  distractor "Tories", :explanation => "They're British"
  distractor "Social Democrats"
end
```

Figura 1.5: Pregunta multirrespuesta con una múltiples respuestas correctas

- **Preguntas de verdadero o falso:** caso particular de las preguntas multirrespuesta de opción única.

```
truefalse 'The earth is flat.', false, :explanation => 'No, just looks that way'
```

Figura 1.6: Pregunta de verdadero o falso

Para todos los tipos de preguntas era posible especificar un comentario opcional que acompañaría al texto de la pregunta.

1.2.1. Código de ejemplo para realizar un cuestionario HTML5

Instalamos la gema:

```
[~]$ gem install ruql
Fetching: ruql-0.0.4.gem (100%)
Successfully installed ruql-0.0.4
1 gem installed
```

Creamos el fichero Ruby que contendrá las preguntas:

```
[~]$ cd tmp
[~/tmp]$ mkdir example
[~/tmp]$ vi example.rb
[~/tmp]$ cat example.rb
```

```
1 quiz 'Example_quiz' do
2
3   fill_in :points => 2 do
4     text 'The capital of California is ____'
5     answer 'sacramento'
6   end
7
8   choice_answer :randomize => true do
9     text "What is the largest US state?"
10    explanation "Not big enough." # for distractors without their own explanation
11    answer 'Alaska'
12    distractor 'Hawaii'
13    distractor 'Texas', :explanation => "That's pretty big, but think colder."
14  end
15
16  select_multiple do
17    text "Which are American political parties?"
18    answer "Democrats"
19    answer "Republicans"
20    answer "Greens", :explanation => "Yes, they're a party!"
21    distractor "Tories", :explanation => "They're British"
22    distractor "Social Democrats"
23  end
24
25  select_multiple do
26    text "Which are American political parties?"
27    answer "Democrats"
28    answer "Republicans"
29    answer "Greens", :explanation => "Yes, they're a party!"
30    distractor "Tories", :explanation => "They're British"
31    distractor "Social Democrats"
32  end
33
34  truefalse 'The week has 7 days.', true
35  truefalse 'The earth is flat.', false, :explanation => 'No, just looks that way'
36 end
```

Para generar el HTML versión imprimible, ejecutamos el siguiente comando:

```
[~/tmp]$ ruql example.rb Html5 > example.html
```

Si deseamos que nuestras preguntas se rendericen usando nuestro propio template, debemos especificarlo con la opción `-t`:

```
[~/tmp]$ ruql example.rb Html5 -t template.html.erb > example.html
```

Las especificaciones de cómo crear nuestro propio template se encuentran en el apartado de Guía de usuario (véase Apéndice B.1).

1.3. Objetivos y actividades a realizar

Los objetivos propuestos para alcanzar en este Trabajo de Fin de Grado ha sido los siguientes:

- Conocer, dominar y practicar con lenguajes y herramientas de desarrollo de aplicaciones web en el **servidor**.
- Conocer, dominar y practicar con diferentes lenguajes y librerías en el **cliente**.
- Conocer, practicar y dominar de herramientas de **desarrollo dirigido por pruebas** (*TDD*) en entornos web.
- Conocer, practicar y dominar diferentes lenguajes de marcas y de estilo.
- Conocer, practicar y dominar diferentes mecanismos de despliegue.
- Conocer, practicar y familiarizarse con diferentes mecanismos de seguridad, autenticación y autorización.
- Conocer, practicar y dominar diferentes herramientas colaborativas y de **control de versiones** (*CVS*).
- Conocer, practicar y dominar **metodologías ágiles** de desarrollo de software.
- Desarrollar una aplicación web para la elaboración y despliegue de cuestionarios.

Y las actividades a realizar en el mismo, tal cual están descritas en la propuesta de **Proyecto de Trabajo de Fin de Grado** firmada por el director y el alumno en la actividad 2 de la asignatura, son las que se describen a continuación:

- Revisión bibliográfica.
- Realización de una aplicación web en la que:
 - Se proporciona soporte mediante una aplicación web a los procesos de evaluación.
 - Se proporciona/extiende un **Lenguaje de Dominio Específico** (*DSL*) para la elaboración de cuestionarios.
 - Se deberá considerar cómo resolver los problemas de seguridad asociados.
 - Redacción de la memoria.
- Preparación de las presentaciones.

1.4. Tecnología usada

Debido a que este Trabajo de Fin de Grado es una extensión de una gema de **Ruby**, se ha utilizado éste como lenguaje de programación.



Además, se ha hecho uso de un numeroso conjunto de gemas y de otras tecnologías enumeradas a continuación:

- HTML5 
- CSS3 
- JavaScript 
- Bootstrap 

- jQuery  write less, do more.
- XRegExp 
- MathJax 
- CodeMirror 
- Mocha 
- Chai 
- Karma 
- Sinatra 
- GitHub 
- Heroku 
- OAuth 2.0 
- Google Drive 

Capítulo 2

Desarrollo

En el capítulo anterior se ha definido el Trabajo de Fin de Grado, especificado los objetivos y actividades a desarrollar y mencionado las tecnologías empleadas para su desarrollo. A continuación, se describirá la metodología de trabajo seguida y se introducirá a los resultados obtenidos para posteriormente describirlos por capítulos de manera detallada.

2.1. Metodología usada

Se ha llevado a cabo una metodología de trabajo *ágil*, común en el campo de la Ingeniería Informática, con reuniones semanales en las que se definían una serie de tareas u objetivos (iteración) y que se presentaban la siguiente semana. De este modo, con la entrega de prototipos funcionales de la aplicación, se han ido testeando, corrigiendo y mejorando las funcionalidades, al mismo tiempo que detectando problemas no contemplados en las fases previas de diseño.

Esta metodología, además, ha propiciado la generación de ideas que se han traducido en nuevas características.

2.1.1. GitHub

Para llevar a cabo esta metodología, se ha usado **GitHub** como herramienta de **Control de Versiones** (CVS). Todo el código implementado se alojaba en dicha herramienta, permitiendo así su cómoda modificación y actualización.

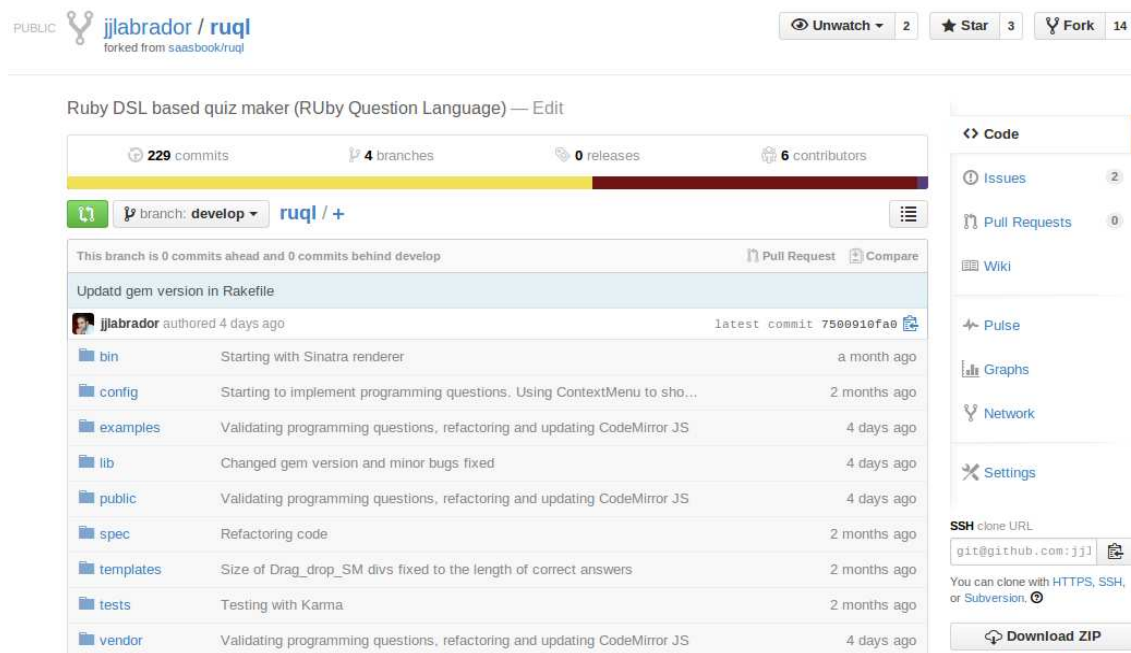


Figura 2.1: Captura del repositorio de la gema en GitHub

El trabajo se dividía en ramas, de modo que la versión estable de la aplicación (*rama master*) quedara aislada de la versión en desarrollo (*rama develop*).

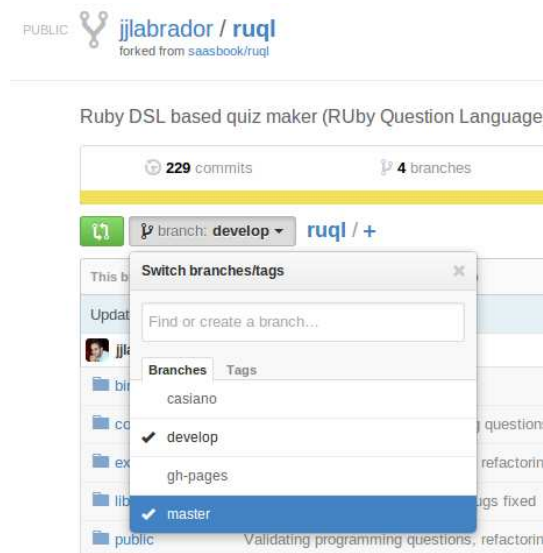


Figura 2.2: Ramas del repositorio propio

GitHub también ha servido para mantener contacto con el creador de la gema e indicarle mi intención de mejorar su gema en mi Trabajo de Fin de Grado. Él ha estado al tanto de mi progreso y tras solicitarle *Pull Requests* con mejoras y correcciones de su gema me ha convertido en colaborador de su repositorio.



Figura 2.3: Pull Request aceptado y cerrado

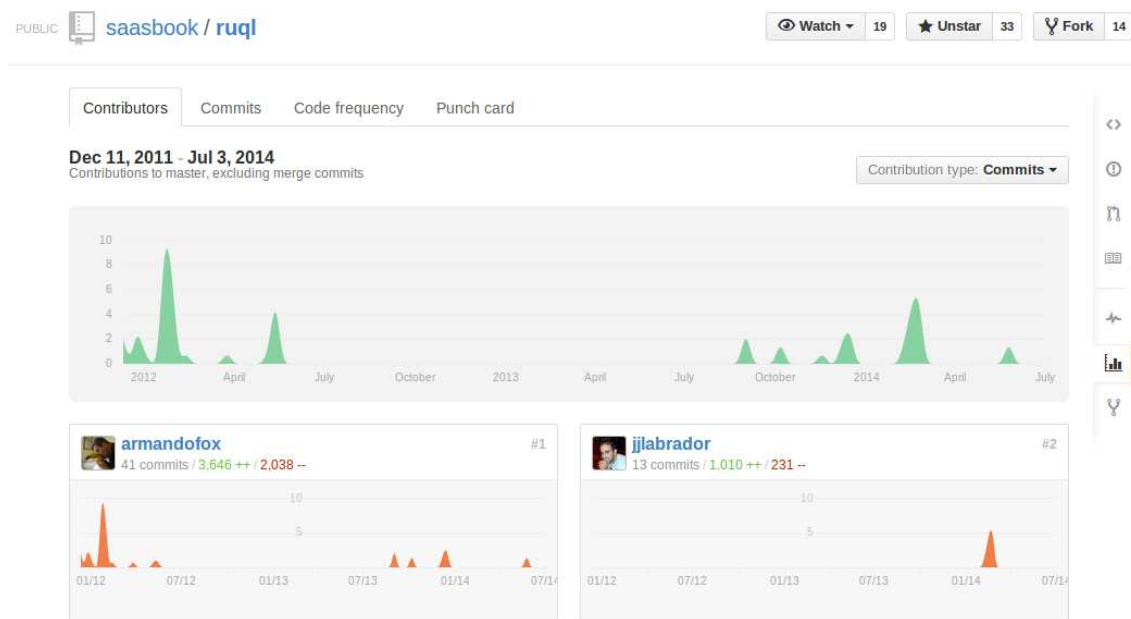


Figura 2.4: Contribuciones hechas al repositorio original

Las nuevas funcionalidades que han ido surgiendo, así como los problemas detectados, se anotaban en el apartado de *issues* con el fin de que quedara constancia de ello y se reflejara el estado en el que se encontraba cada uno.

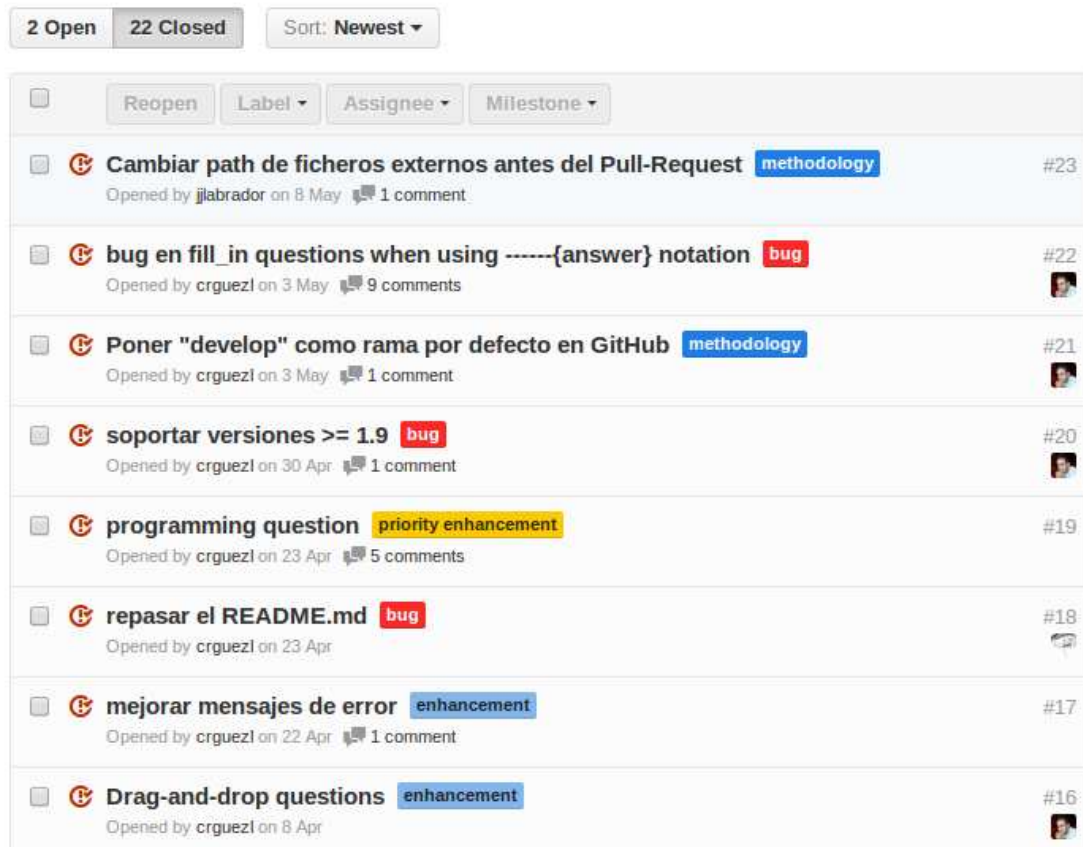


Figura 2.5: Apartado de issues cerrados

2.1.2. Testing

Dentro de la metodología también ha habido etapas de testing, haciendo uso de la metodología de Desarrollo Dirigido por Pruebas **TDD** (Test Driven Development). Se ha empleado la herramienta *Spec* para los test en Ruby y se han usado los frameworks *Mocha*, *Chai* y *Karma* para los test del HTML y el JavaScript.

2.1.3. Experiencia de usuario

Por otra parte, el tutor del Trabajo de Fin de Grado ha hecho pruebas reales usando los prototipos de la aplicación con alumnos de sus asignaturas. De este modo, se comprobaba el funcionamiento de la aplicación en un entorno real y se recibía un valioso feedback para mejorar en las siguientes iteraciones.

2.2. Resultados

Tras el desarrollo del Trabajo de Fin de Grado, se distinguen tres claros resultados: por un lado tenemos la **corrección de errores y mejoras de la gema original**. En segunda instancia, contamos con el renderer **HtmlForm**, válido para realizar una autoevaluación por parte del alumnado que le sirve, a modo de entrenamiento, para afrontar el examen final. Por último, contamos con el renderer **Sinatra**, que genera una aplicación Sinatra con todo lo necesario para su despliegue y puesta en funcionamiento.

Se explicarán cada uno de estos resultados en los siguientes capítulos de la memoria.

Capítulo 3

Mejoras del DSL original

Fruto del estudio del código y de ejecuciones sucesivas de la gema se detectaron una serie de errores de funcionamiento, por lo que antes de implementar mis mejoras propias era conveniente solucionar los problemas existentes. Además, se añadieron algunos cambios para mejorar el funcionamiento de la gema. A continuación se detallan las mejoras realizadas:

- Corrección de errores en el funcionamiento de la gema. Las opciones enumeradas a continuación no funcionaban correctamente:
 - La opción que permite indicar si el orden de las respuestas en las preguntas de completar espacios en blanco importa o no.
 - La opción de añadir comentarios opcionales a los textos de las preguntas.
 - La opción *raw* que permite incrustar el texto de las preguntas entre etiquetas `<pre>` HTML.
 - La opción de explicación global para todos los *distractors* no funcionaba.
- Refactorización de código, evitando la repetición del mismo en la medida de lo posible.
- Añadido manejo de excepciones tras errores de ejecución.
- Añadida la opción en línea de comandos *-version* para comprobar la versión de la gema.
- Añadida la opción en línea de comandos *-help* para ver la ayuda.

3.1. Problemas encontrados y soluciones

A continuación se detallan los problemas encontrados durante la implementación de las mejoras del DSL original y las soluciones encontradas para los mismos.

3.1.1. Entender el funcionamiento del código de la gema

Solución

Leer la documentación de la gema, generar cuestionarios de pruebas y estudiar el código fuente.

3.1.2. Corregir tests y funcionalidades de la gema

Solución

Tras realizar el correspondiente *fork* en GitHub para empezar a implementar mis modificaciones, ejecuté los tests de la gema original para comprobar la ausencia de fallos. Al finalizar, algunos tests fallaron por lo que decidí corregirlos. Del mismo modo, algunas gemas de testing existentes en el Gemfile presentaban incompatibilidades con las nuevas versiones de Ruby, por lo que también se corrigió.

Capítulo 4

HtmlForm renderer

Este renderer permitirá generar un documento HTML5 con un formulario en el que se encuentran todas las preguntas listas para ser completadas desde el navegador. Posee además todos los JavaScripts necesarios para la corrección automática de las mismas.

El objetivo de este renderer es proporcionar al alumno una serie de preguntas para que practique con vistas a afrontar un examen oficial. La principal ventaja que ofrece es la portabilidad: al ser un documento HTML puede alojarse en cualquier servidor o ejecutarse localmente desde un navegador sin tener que configurar nada previamente. Solo es necesario tener conexión a Internet ya que existe un JavaScript que necesita ser descargado mediante un **CDN**.

A continuación se enumerarán todas las características de este renderer:

- Permite añadir uno o más JavaScripts al cuestionario que se generará.

```
[~/tmp]$ ruql example.rb Html5 -j examples/test.js > example.html
```

- Permite añadir uno o más ficheros de fragmentos de código HTML en la cabecera del cuestionario que se generará.

```
[~/tmp]$ ruql example.rb Html5 -h examples/partial.html > example.html
```

- Permite la posibilidad de añadir más de una hoja de estilo CSS al cuestionario que se generará.

```
[~/tmp]$ ruql example.rb Html5 -c examples/style1.css
-c examples/style2.css > example.html
```

- Permite añadir un header y un footer personalizado al cuestionario que se generará. Se deberá indicar en el fichero Ruby. Se puede especificar como un *string* o indicar el path donde se encuentra el fichero que contiene dicho código HTML.

```
head : 'examples/header.html'
```

Figura 4.1: Código para incluir un header personalizado

```
foot : 'examples/footer.html'
```

Figura 4.2: Código para incluir un footer personalizado

- Los textos de las preguntas admiten ahora caracteres HTML escapados usando el método *escape*.

```
tag = '<a href="www.google.es"></a> '

fill_in do
  text "<i>Example of escaped HTML </i><br> #{escape(tag)}" + "is a ---- "
  answer /^link$/
end
```

Figura 4.3: Ejemplo de pregunta con HTML escapado

- El cuestionario es capaz de renderizar expresiones escritas en **LaTeX**.

```
fill_in do
  text %Q{
    Calculate the determinant of this matrix:
    $$\mathbf{A} = \begin{vmatrix}
    1 & 3 \\
    2 & 4
    \end{vmatrix}$$
    <br/>
    ----
  }
  answer -2
end
```

Figura 4.4: Ejemplo de pregunta con código LaTeX

[1 point] Calculate the determinant of this matrix:

$$\mathbf{A} = \begin{vmatrix} 1 & 3 \\ 2 & 4 \end{vmatrix}$$

Figura 4.5: Ejemplo de pregunta con código LaTeX renderizada

- Las preguntas de completar permiten ahora respuestas numéricas y de código JavaScript.

```
fill_in do
  text %Q{
    Solve:
    
$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

    when n = 5 and k = 2:
    ----
  }
  explanation "Not exactly"
  answer 10
  distractor 9
end
```

Figura 4.6: Ejemplo de pregunta con respuesta numérica

```
fill_in do
  text %Q{
    Diga dos números x = ---- e y = ---- que multiplicados den 100
  }
  answer JavaScript.new(%q{result = function(x,y) { return (x * y === 100); }})
end
```

Figura 4.7: Ejemplo de pregunta con respuesta de tipo JavaScript

- Las espacios para rellenar las respuestas de las preguntas de completar se ajustan al tamaño de dicha respuesta. Para ello basta especificar tantos guiones '-' como caracteres contenga la respuesta en el fichero Ruby. Es necesario un mínimo de tres guiones, de lo contrario, no se traducirán dichos guiones a etiquetas input HTML. En caso de querer mostrar mas de tres guiones sin que sean traducidos a inputs, se deben escapar usando un *backslash* ('\\')

```
fill_in do
  text "\- \- \- -The capital of Tenerife is ----- Cruz de ----- \- \- \- -"
  answer [/Santa/i, /Tenerife/i]
end
```

Figura 4.8: Ejemplo de pregunta con respuestas de múltiples longitudes

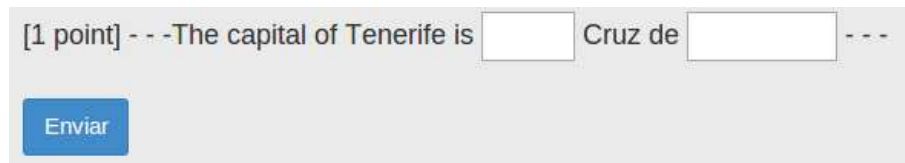


Figura 4.9: Ejemplo de pregunta con respuestas de múltiples longitudes renderizada

- Cuando en una pregunta existen numerosos huecos para escribir respuestas, en el array de respuestas podrá haber alguna de ellas que no mantiene su correspondencia con el índice que ocupa, por lo que se han añadido dos nuevas maneras simplificadas de escribir estas preguntas de completar:
 - Colocando la respuesta de la pregunta junto a sus guiones correspondiente. Para este tipo de escritura, solo se admiten respuestas de tipo *String*.

```
fill_in do
  text "The three stooges are -----{larry}, ----{moe}, and -----{curly}."
end
```

Figura 4.10: Ejemplo del primer tipo de pregunta simplificada

- Usando una notación de *Hash*: especificando una clave seguida de los guiones de la respuesta y definir su correspondencia en el método definido para escribir la respuesta.

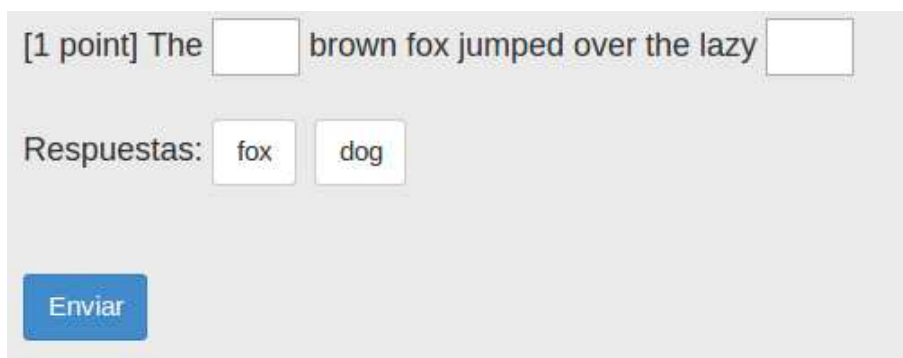
```
fill_in do
  text "The capital of Tenerife is -----{:santa} Cruz de -----{:tenerife}"
  answer :santa => /Santa/i, :tenerife => /Tenerife/i
end
```

Figura 4.11: Ejemplo del segundo tipo de pregunta simplificada

- Se han añadido dos nuevos tipos de preguntas:
 - Preguntas de **Drag and Drop**: para preguntas de completar y preguntas tipo test (de respuesta única y multirrespuesta).

```
drag_drop_fill_in do
  text "The ---- brown fox jumped over the lazy ----"
  answer ['fox', 'dog']
end
```

Figura 4.12: Ejemplo de pregunta de completar con Drag and Drop



[1 point] The brown fox jumped over the lazy

Respuestas:

Figura 4.13: Ejemplo de pregunta de completar con Drag and Drop renderizada

```
drag_drop_choice_answer do
  text "Relate these concepts"
  relation :Facebook => 'Mark Zuckerberg', :Twitter => 'Jack Dorsey'
end
```

Figura 4.14: Ejemplo de pregunta tipo test de respuesta única con Drag and Drop



The screenshot shows a web-based interface for a drag-and-drop question. At the top, it says "[1 point] Relate these concepts". Below this, there are two columns of draggable items. The left column contains two boxes labeled "Facebook" and "Twitter". The right column contains two boxes labeled "Mark Zuckerberg" and "Jack Dorsey". In the center, there are two empty rectangular boxes for the user to place the items. At the bottom left, there is a blue button labeled "Enviar".

Figura 4.15: Ejemplo de pregunta tipo test de respuesta única con Drag and Drop renderizada

```
drag_drop_select_multiple do
  text "Relate these concepts"
  relation :Ruby => ['Sinatra', 'Rails'], :JavaScript => 'jQuery'
end
```

Figura 4.16: Ejemplo de pregunta de tipo test multirrespuesta con Drag and Drop

Figura 4.17: Ejemplo de pregunta de tipo test multirrespuesta con Drag and Drop renderizada

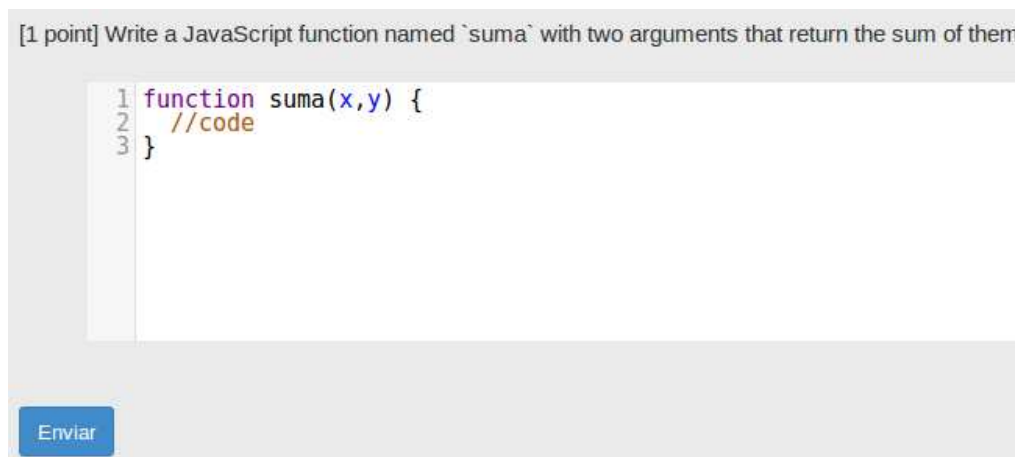
NOTA: en este último tipo de pregunta Drag and Drop, la longitud del contenedor donde se insertan las respuestas es el máximo de la suma de las longitudes de las respuestas correctas.

- Preguntas de **programación**:
 - Este tipo de preguntas sólo admite código JavaScript debido a que la corrección de preguntas también tiene lugar en el navegador cliente.
 - La respuesta asignada a este tipo de preguntas debe ser un código JavaScript que valide la respuesta introducida por el alumno. Este código puede escribirse en notación de *string* o especificar el path donde se encuentra el fichero que contiene dicho código.

```
programming :language => 'JavaScript', :height => 150, :width => 800 do
  text %Q{Write a JavaScript function named `suma` with two arguments that return the sum of them}
  answer JavaScript.new(:'examples/test_suma.js')
end
```

Figura 4.18: Ejemplo de pregunta de programación

- Se creará un *textarea* de unas dimensiones definidas por defecto que también se pueden personalizar y se coloreará el código escrito para facilitar su lectura.



[1 point] Write a JavaScript function named `suma` with two arguments that return the sum of them

```
1 function suma(x,y) {  
2   //code  
3 }
```

Enviar

Figura 4.19: Ejemplo de pregunta de programación renderizada

- Validación automática mediante JavaScript de las preguntas que han sido rellenas. Para corregir respuestas de tipo *Regex* se utiliza **XRegExp**, mucho más potente que las expresiones regulares proporcionadas en JavaScript nativo. Esta validación muestra:
 1. La nota obtenida en esa pregunta.
 2. La explicación asociada a esa respuesta (si se ha especificado en el fichero Ruby).



[1 point] What is the largest US state?

☐ Alaska

☒ Texas **Incorrecto - That's pretty big, but think colder.**

☐ Hawaii

0.00/1.00 puntos

Mostrar respuesta Enviar

Figura 4.20: Ejemplo de pregunta corregida

3. La puntuación total al final de la página.



Figura 4.21: Puntuación total

- Almacenamiento local de las respuestas introducidas usando **Local Storage** de HTML5.
- Menú contextual al hacer click derecho sobre el campo de respuesta para ver la respuesta correcta de dicha pregunta. Esta funcionalidad sólo está disponible para preguntas de completar, cuyas respuestas sean *strings*, *regexps* o numéricas.



Figura 4.22: Ejemplo de mostrar respuesta correcta en pregunta de completar

Para las preguntas tipo test existe un botón que marca las respuestas correctas.

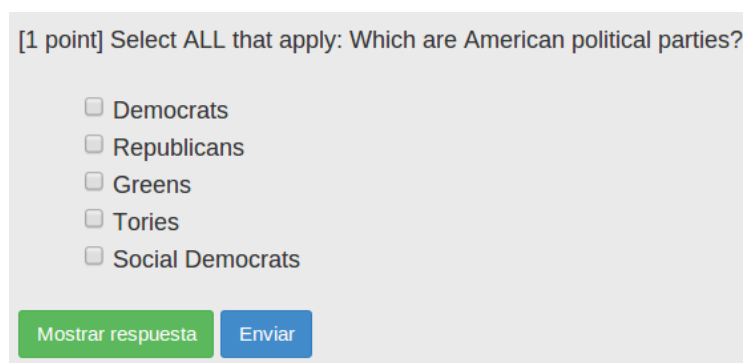


Figura 4.23: Ejemplo de mostrar respuesta correcta en pregunta tipo test

- Internacionalización: la gema comprueba el idioma del sistema para ofrecer la traducción adecuada al idioma del usuario de diversos mensajes como la corrección de cada pregunta (correcto/incorrecto) o la alerta que anuncia que se ha borrado el Local Storage. Actualmente solo soporta inglés y español. Para cualquier otro idioma, se utiliza el inglés por defecto.

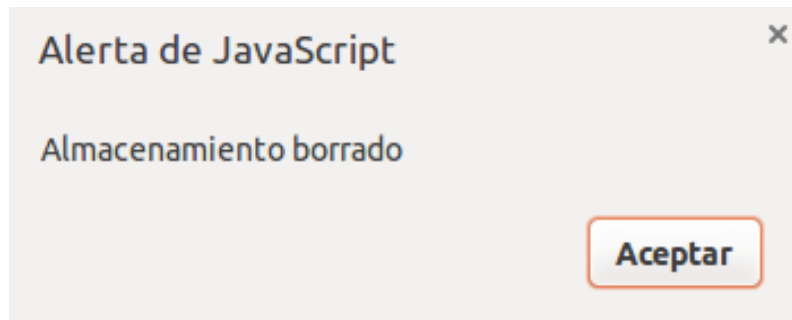


Figura 4.24: Mensaje en español de Local Storage borrado

4.1. Problemas encontrados y soluciones

4.1.1. Corrección de preguntas de Ruby en JavaScript

Se ha tratado de usar **Opal** para traducir preguntas escritas en Ruby a JavaScript y poder validarlas desde el navegador. Se consiguió en parte, pues no fue posible obtener el código de una *Lambda* o un *Proc*, ya que Ruby los evaluaba antes de llegar al JavaScript.

Solución

Permitir solamente preguntas de código JavaScript en este renderer.

4.1.2. Alojamiento librería MathJax en un directorio de la gema

Todos los JavaScript y CSS de terceras partes se encuentran alojadas en un directorio denominado *vendor*. Para los JavaScripts y CSS propios existe otro directorio llamado *public*. Estos ficheros se insertan en el HTML generado, de modo que no es necesario manejar varios ficheros junto con el HTML.

Sin embargo, el framework **MathJax**, usado para insertar texto en LaTeX en el HTML, no se ha podido insertar en el mismo. Consta de un amplio número de

ficheros y cada uno de ellos es una dependencia de otro. Además, existen numerosas imágenes por lo que no ha sido posible insertarlo en el HTML de salida.

Solución

Hacer uso de un **CDN** para utilizar este framework.

Capítulo 5

Sinatra renderer

Este otro renderer genera una aplicación Sinatra con todo lo necesario para ser desplegada en **Heroku** o ejecutar localmente. Se hace uso de **Google Drive** para almacenar una copia del cuestionario y para alojar las preguntas y respuestas de los alumnos.

Para usar este renderer es necesario añadir especificar algunos parámetros más en el fichero Ruby que contiene el cuestionario. Estos parámetros se enumeran a continuación:

1. La dirección de correo en **Gmail** del profesor. Es posible especificar más de una dirección usando una notación de *Array*.

```
teachers "jjlabradorglez@gmail.com"
```

Figura 5.1: Método para especificar los profesores permitidos en la aplicación

2. El path de un fichero **CSV** con los datos de los alumnos.

```
students : "examples/students.csv"
```

Figura 5.2: Método para especificar los alumnos permitidos en la aplicación

3. El path de un fichero de configuración denominado *config.yml* que contiene información del cuestionario.

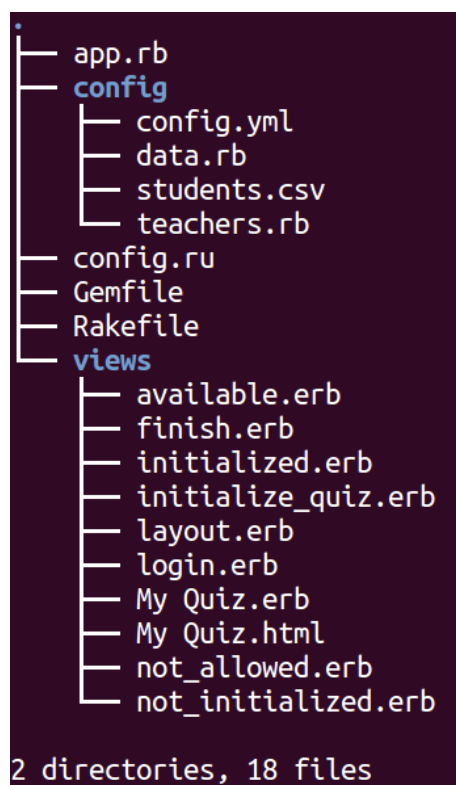
```
config : "examples/config.yml"
```

Figura 5.3: Método para especificar la configuración de la aplicación

La sintaxis para ejecutar este renderer es la siguiente:

```
[~/tmp]$ ruql example.rb Sinatra -t templates/htmlform.html.erb
```

Tras su ejecución, se creará una carpeta llamada *app* en el directorio donde nos encontremos. Este directorio contendrá lo siguiente:



```
.
├── app.rb
├── config
│   ├── config.yml
│   ├── data.rb
│   ├── students.csv
│   └── teachers.rb
├── config.ru
├── Gemfile
├── Rakefile
├── views
│   ├── available.erb
│   ├── finish.erb
│   ├── initialized.erb
│   ├── initialize_quiz.erb
│   ├── layout.erb
│   ├── login.erb
│   ├── My Quiz.erb
│   ├── My Quiz.html
│   ├── not_allowed.erb
│   └── not_initialized.erb
└── 2 directories, 18 files
```

Figura 5.4: Listado de ficheros y directorios generado dentro del directorio *app*

Para saber cómo rellenar los ficheros de entrada necesario y obtener información de los directorios y ficheros generados, véase Apéndice B.1.

La aplicación resultante funciona del siguiente modo:

1. Una vez ejecutada, el profesor deberá autenticarse con su cuenta de Gmail y activar el cuestionario. Hecho esto, se creará en su **Google Drive** una carpeta que contendrá una copia de examen y una hoja de cálculo con las preguntas y respuestas correctas del cuestionario. También guardará la información de los usuarios.
2. Una vez activado, los alumnos podrán acceder a realizar el mismo. Cuando lo finalicen, se generará una copia de su examen en la carpeta de Google Drive del profesor y se escribirá automáticamente su hoja de cálculo para añadir la nota que ha sacado el alumno individualmente en cada pregunta y de manera global. De este modo, quedará constancia de la realización del mismo.
3. Los alumnos podrán reintentar el cuestionario todas las veces que deseen mientras se encuentre activo. Éste dejará de estar activo cuando finalice la fecha límite establecida por el profesor en el fichero de configuración. Al reintentar el cuestionario, se actualizará Google Drive con las nuevas calificaciones del alumno.

NOTA: será responsabilidad del profesor facilitar la nota a los alumnos en el momento que estime oportuno.

Para resolver el gran problema de la autenticación de usuarios, se hace uso de OAuth. De este modo, delegamos todo el servicio a Google y evitamos, por tanto, posibles brechas de seguridad que den lugar a suplantaciones de identidad o exposición de datos sensibles de los usuarios a terceras personas.

5.1. Problemas encontrados y soluciones

5.1.1. Timeout corto entre peticiones del navegador al servidor

Al inicializar el cuestionario, el servidor tarda en responder ya que es necesario escribir en Google Drive una cantidad de datos considerable. Cuando termina la escritura, nos debe mostrar una vista en la que se puede comprobar que el cuestionario ha sido inicializado correctamente. Esta vista sólo se muestra una vez y únicamente al profesor que inicializó el cuestionario. El resto de ocasiones, se nos redirige al cuestionario en sí. Sin embargo, mientras se está escribiendo en Google Drive el navegador interpreta que el servidor tarda en responder, por lo que le manda una

nueva petición y éste la despacha en cuanto finaliza la escritura en Google Drive. El problema reside en que atiende a esta nueva petición y, al ver que el cuestionario ya ha sido inicializado, nos redirige al cuestionario, saltándose la vista que debería aparecer la primera vez.

Solución

Usar como servidor **WeBrick** en lugar del que ejecuta Sinatra por defecto (**Thin**). WeBrick no permite múltiples peticiones por lo que se nos muestra la vista adecuada al inicializar el cuestionario.

5.1.2. Lugar de almacenamiento de las respuestas de los alumnos

Estaba claro que la información de los cuestionarios había que guardarla en algún lado. La idea principal era almacenar todas las preguntas y respuestas de los alumnos en una base de datos, pero preocupaba el hecho de que ésta se alojara en un servidor desconocido y que, por algún motivo, se perdiera dicha información sensible.

Solución

Viendo la evolución que ha tenido la herramienta Google Drive y el aumento considerable de su uso por parte de docentes, decidí sustituir las tradicionales y siempre monótonas consultas a bases de datos por esta herramienta de almacenamiento que permite visualizar y administrar fácilmente toda la información. Además de ser segura, la información reside en la cuenta del profesor y éste la puede exportar donde desee cómodamente.

5.1.3. Problema de seguridad al evaluar código Ruby en el servidor

En esta primera versión del renderer, la aplicación generada evalúa el código introducido por el alumno sin ningún mecanismo de protección, por lo que introduciendo algún script comprometedor en el campo de alguna respuesta se podría comprometer la información de la aplicación.

Este es un aspecto importante a tratar en próximas mejoras del renderer.

Capítulo 6

Conclusiones y trabajos futuros

Este capítulo es obligatorio. Toda memoria de Trabajo de Fin de Grado debe incluir unas conclusiones y unas líneas de trabajo futuro

Capítulo 7

Summary and Conclusions

This chapter is compulsory. The memory should include an extended summary and conclusions in english.

7.1. First Section

Capítulo 8

Presupuesto

Este capítulo es obligatorio. Toda memoria de Trabajo de Fin de Grado debe incluir un presupuesto.

8.1. Sección Uno

Tipos	Descripcion
AAAA	BBBB
CCCC	DDDD
EEEE	FFFF
GGGG	HHHH

Tabla 8.1: Tabla resumen de los Tipos

Apéndice A

Glosario de términos

A.1. T

Términos + enlaces

Lenguaje de Dominio Específico (DSL)

Ruby

Sinatra

RuQL

Bootstrap

Open EdX

Renderer

Fork

Gema

HTML5

CSS3

Desarrollo Dirigido por Test (TDD)

Control de Versiones (CVS)

Metodología ágil

GitHub

Template

Gemfile

Header

Footer

CSV

ERB

World Wide Web

Web semantica

AJAX

Moodle

Gamificacion

Cliente-Servidor

TDD

Framework

CDN

Lambda, Proc

Opal

WeBrick

Thin

Apéndice B

Guía de usuario final

B.1. Guía de usuario: Seccion 1

Texto

Bibliografía

- [1] C. Darwin, *The Origin Of Species*. November 1859.
- [2] “ACM LaTeX Style.” http://www.acm.org/publications/latex_style/.
- [3] D. H. Bailey and P. Swarztrauber, “The fractional Fourier transform and applications,” *SIAM Rev.*, vol. 33, no. 3, pp. 389–404, 1991.
- [4] A. Bayliss, C. I. Goldstein, and E. Turkel, “An iterative method for the Helmholtz equation,” *J. Comp. Phys.*, vol. 49, pp. 443–457, 1983.
- [5] “FACOM OS IV SSL II USER’S GUIDE, 99SP0050E5,” tech. rep., 1990.
- [6] C. Goldstein, “Multigrid methods for elliptic problems in unbounded domains,” *SIAM J. Numer. Anal.*, vol. 30, pp. 159–183, 1993.
- [7] P. Swarztrauber, *Vectorizing the FFTs*. Academic Press, New York, 1982.
- [8] S. Taásan, *Multigrid Methods for Highly Oscillatory Problems*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1984.