UNIVERSIDAD
REY JUAN CARLOS

DOBLE TITULACIÓN EN INGENIERÍA DE TELECOMUNICACIÓN Y LICENCIATURA EN ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS

Curso Académico 2015/2016

Trabajo Fin de Carrera

DR. SCRATCH: FOMENTANDO LA CREATIVIDAD Y VOCACIÓN CIENTÍFICA CON SCRATCH

Autor : Mari Luz Aguado Jiménez

Tutor : Dr. Gregorio Robles Martínez

Co-Tutor : Jesús Moreno León

Proyecto Fin de Carrera

DR. SCRATCH: FOMENTANDO LA CREATIVIDAD
Y VOCACIÓN CIENTÍFICA CON SCRATCH

Autor : Mari Luz Aguado Jiménez

Tutor : Dr. Gregorio Robles Martínez

Co-Tutor : Jesús Moreno León

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 20XX, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 20XX

Dedicado a

mi familia, pareja y amigos.

II

Agradecimientos

—-(Aquí vienen los agradecimientos. . . Aunque está bien acordarse de la pareja, no hay que olvidarse de dar las gracias a tu madre, que aunque a veces no lo parezca disfrutará tanto de tus logros como tú. . . Además, la pareja quizás no sea para siempre, pero tu madre sí.)

Resumen

Dr. Scratch es una plataforma web de software libre que permite analizar proyectos realizados en *Scratch* -lenguaje orientado a la enseñanza- aportando feedback sobre determinados aspectos y aptitudes relacionadas con el *Pensamiento Computacional*. El **objetivo** principal del proyecto es dar soporte tanto a maestros como estudiantes de educación obligatoria en sus primeros pasos en la programación con Scratch.

Este **proyecto** es el resultado del trabajo de cinco personas, no sólo el mío. Por lo que iré indicando durante toda la memoria cuál ha sido mi aportación de la forma más concreta posible. Destacar, que es un proyecto amplio, ya que ha sido desarrollado durante algo más de un año y aún está en proceso.

Las **tecnologías** utilizadas han sido diversas debido a que es un proyecto en producción, hay miles de proyectos analizados por numerosos colegios y organizaciones. Para conseguir tener la web en producción hemos utilizado Azure y la licencia Apache. Por debajo de ello, tenemos un servidor desarrollado en Python 2.7 con la ayuda del framework Django 1.8 que nos ha facilitado multitud de tareas de la parte back-end. Además, para el registro de usuarios, organizaciones y guardar toda la información de los proyectos analizados tenemos una base de datos en MySQL. Pero el front-end es lo que más trabajo nos ha supuesto, porque hemos ido realizando modificaciones según las necesidades de los usuarios. De este modo, HTML, CSS con la ayuda de Bootstrap, Javascript y AJAX han sido utilizados diariamente.

Actualmente, se está introduciendo la programación en las aulas de numerosos colegios de nuestro país. Pero existe la **problemática** de falta de formación en esta materia de la mayoría de maestros de educación obligatoria. El proyecto trata de suplir esta deficiencia y guiar a dicho colectivo en sus primeros pasos. Además, sus dashboards han sido diseñados de un modo tan sencillo, que la idea es que un alumno con un nivel de comprensión lectora media, sea capaz de aprender por sí mismo utilizando la herramienta.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La forma más eficaz de interesar a una persona por un tema, es hacer que se **involucre**. Y las formas de conseguir ese compromiso, dependen en gran medida, de nuestras aptitudes personales, tales como la capacidad de comunicación, creatividad, innovación y liderazgo. *Pero hoy en día tenemos un gran aliado, las nuevas tecnologías*.

En este capítulo, se trata la situación actual de la educación y cómo aprender a programar en *Scratch*, puede ayudar a desarrollar aptitudes relacionadas con el *Pensamiento Computacional*.

1.1. Marco general

Uno de los sectores más gratificantes en los que se puede trabajar, es la **educación**. Pero en sí mismo, es a la vez, todo un reto. Por descontado, decir, que no todo estudiante es igual, pero además, los estudiantes cambian junto con la *evolución de la sociedad* y hoy en día la sociedad está cambiando a un ritmo desproporcionado, ligada al *desarrollo de la tecnología*. Ya no hacemos -casi- nada como hace veinte años, y por tanto, tampoco aprendemos de la misma forma.

Como siempre, en la evolución del ser humano, lo único que está seguro es el **cambio**, y es nuestra capacidad de adaptación a dichos cambios, lo que nos hace tener éxito, o no, en los retos que nos proponemos. La educación no iba a ser diferente y en los últimos años, los currículos españoles han sufrido inmensas modificaciones con el *proceso de Bolonia*. Ahora toca ponerse manos a la obra, para tratar de adaptarnos a lo que la sociedad está demandando, para poder seguir dando una educación de calidad. Es el momento de incorporar *nuevas herramientas* y

metodologías, que permitan desarrollar las habilidades que el mercado laboral está demandando.

En unos años, la mayoría de las ofertas de trabajo serán para puestos cualificados, en los que se demandará -prácticamente en todos- tener nociones de **programación**. La programación, esa gran desconocida, que muchas veces tiene connotaciones negativas, sobre todo, entre la sección femenina.

Pero el mayor problema, es el citado. *No se conoce.* No se ha tratado de involucrar a la sociedad en dicha materia, hasta ahora que el día a día ha empezado a exigirlo. Todo funciona a través de internet, si no tienes tu comercio en la red estas anticuado, así que necesitas aprender sobre e-commerce. Y necesitas una web en la que se muestren tus productos, la gente no tiene tanto tiempo para desplazarse, o si lo tiene, ¿para qué ir a tu tienda si puede ver los productos de otro desde su casa? Necesitas alguien que te programe una página web propia.

Y este es el caso más sencillo, ya que la programación está tomando también un gran protagonismo en muchísimos otros sectores, tales como la industria, muchísimas fábricas poseen robots programados. Así pues, se está convirtiendo en una *tarea obligatoria a la hora de entrar en el mercado laboral.*

Pero, **¿y sí empezamos a enseñar a programar desde la infancia?** Numerosos estudios están demostrando que aprendiendo a programar se desarrolla el *Pensamiento Computacional*, el cual consta de ciertas habilidades, tales como la abstracción o lógica. Estas, no únicamente son útiles para aprender a programar, sino en muchísimas de las demás asignaturas del currículo pueden servir para involucrar al alumnado en un proyecto que puede tocar y modificar a su gusto a través de la programación. *Puede aprender mientras experimenta*.

Un ejemplo claro es el de la abstracción, que se ve directamente relacionada con las matemáticas, si se enseña a dividir un problema grande en otros más pequeños, es más fácil reconocer cada parte del problema. Así, el alumno sabrá identificar cuándo tiene que sumar o cuándo multiplicar, cuando tiene un problema con varias operaciones.

Hace años, aprender a programar era una tarea compleja, la información para empezar a hacer tu propio programa, sólo se podía encontrar en libros complicadísimos de entender. Y los lenguajes de programación que existían no eran los más intuitivos. Hoy en día, no hay excusas.

En la siguiente subsección, se va a presentar un lenguaje de programación muy apropiado para iniciarse en el mundo de la programación, de un modo simple, motivador y divertido.

1.2. Programación con Scratch

Scratch[1] es un lenguaje de programación orientado a la enseñanza. Ha sido diseñado por el *Grupo Lifelong Kindergarten del Laboratorio de Medios del MIT*, y dirigido por *Mitchel Resnick*. Su principal **objetivo**, es ayudar a los más jóvenes a pensar de forma creativa, razonar sistemáticamente, y trabajar de forma colaborativa. Aptitudes tan necesarias para la vida en el siglo XXI.

La mayor **fortaleza** de Scratch, es permitir crear historias, juegos y animaciones *sin tener conocimientos previos de programación*. La forma de programar en este lenguaje es uniendo piezas estilo *LEGO*.

Figura 1.1: Logo de Scratch

No es necesario escribir código, con su respectiva sintaxis y todo lo que ello conlleva. En lugar de utilizar tiempo en aprender los detalles específicos que tiene cada lenguaje de programación, Scratch tiene una serie de **bloques** agrupados por categorías -cada una con un color, para que sean claramente identificables- dentro de la cual tenemos las piezas del puzzle que arrastraremos para ir indicándole a nuestro videojuego o historia, cuáles son las instrucciones a seguir. En la *Figura 1.2* podemos observar cuáles son los principales bloques que pueden ser utilizados en Scratch y apreciar todo su potencial.

[1] `https://scratch.mit.edu/`

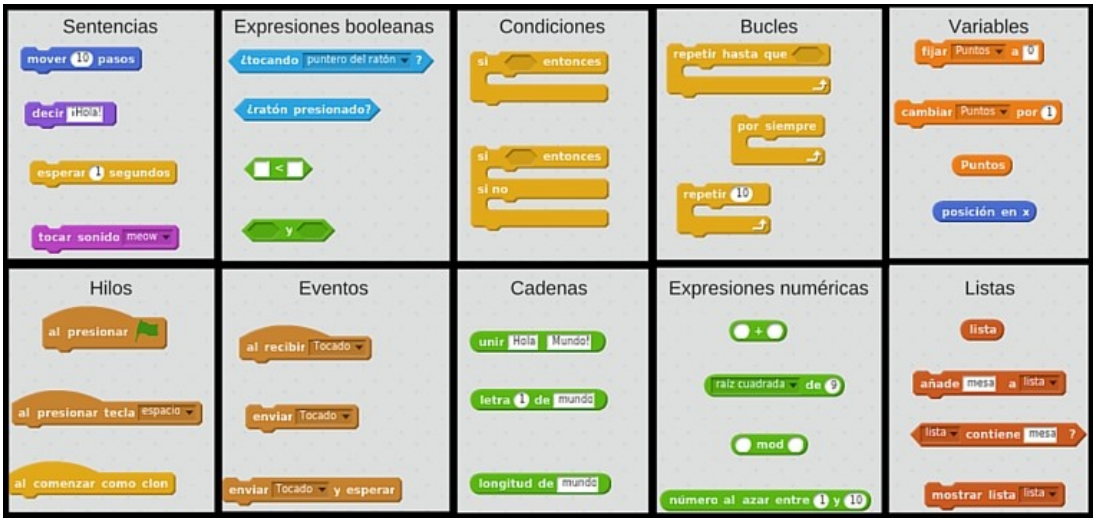

Figura 1.2: Bloques en Scratch

Pero no se queda ahí, Scratch, es además, una **comunidad** donde se pueden *compartir* los proyectos realizados y se pueden exponer dudas en su *foro*. De esta forma, se puede aprender de forma dinámica con la ayuda de otros programadores y observando lo realizado en otros proyectos. También tenemos la opción de *reinventar* otros proyectos, es decir, coger un proyecto realizado por otro Scratcher y modificarlo a nuestro gusto.

Figura 1.3: Estadísticas de la comunidad de Scratch

En la *Figura 1.3* se muestran las estadísticas de la comunidad de Scratch en enero de 2016.

Este lenguaje de programación es *utilizado en más de 150 países y en los más diversos entornos* -museos, bibliotecas, escuelas o universidades son algunos- principalmente, por jóvenes entre los 8 y 16 años. El único requisito para utilizarlo es saber leer.

Actualmente la versión es Scratch 2.0 pero hubo una versión anterior (1.4) que aún está siendo utilizada, sobre todo en su **versión offline**. Porque pensando en las dificultades de conexión

que se tiene en algunos centros, Scratch posee una versión descargable que puede ser utilizada sin conexión a la red.

Scratch se ha convertido en un movimiento mundial con su propio día -Scratch Day- y una conferencia que se realiza cada verano -Scratch Conference- en la que el equipo de Dr. Scratch participó en 2015. Se organizó en Ámsterdam, donde pudimos observar las tendencias e ideas mundiales desarrolladas para potenciar las habilidades de los jóvenes programadores que utilizan Scratch.

1.3. Marco de referencia

Dr. Scratch[2] surge de la observación de mi tutor en este proyecto -Dr. Gregorio Robles- y del doctorando -Jesús Moreno- de la necesidad de una herramienta para Scratch que otorgue feedback sobre los diferentes aspectos que engloban el *Pensamiento Computacional*.

Tras años dedicados a la enseñanza de programación y ciencias de la computación, llegaron a la siguiente conclusión; era necesario tener una herramienta similiar a Pylint -herramienta que permite *detectar malas prácticas en el código del lenguaje de programación* Python- para el nuevo lenguaje de programación Scratch.

Para poder analizar un proyecto programado con Scratch necesitamos inspeccionar su archivo. Los archivos generados en Scratch para cada proyecto, tienen un formato comprimido concreto con extensión *.sb2* cuyo contenido está conformado por las distintas imágenes y sonidos utilizados en el proyecto y un *fichero JSON que contiene la información relevante* a los bloques de Scratch que se han utilizado. Es decir, para evaluar cómo de bien se ha programado un proyecto, lo que nos interesa es conocer dicho JSON y analizar su contenido.

La tarea de analizar toda la información otorgada por el fichero JSON no es fácil, pero mis tutores encontraron en Github[3] un plugin que ya hacía dicha tarea. Este plugin es **Hairball**[4] que ha sido implementado por *Bryce Boe*[5] y presentaba información sobre **los aspectos del Pensamiento Computacional**: *pensamiento lógico, control de flujo, abstracción, paralelismo, interactividad con el usuario, representación de la información y sincronización*. Y además

[2] `http://www.drscratch.org/`
[3] `https://github.com/`
[4] `https://github.com/ucsb-cs-education/hairball`
[5] `https://github.com/bboe`

también tenía en cuenta dos malos hábitos de programación: **código muerto y atributos no inicializados**.

Mis tutores incluyeron otros dos plugins a *Hairball* que inspeccionaban el fichero JSON buscando indicios de otros dos de los **malos hábitos que todo programador novato tiene al principio**: *programas duplicados y nombres inadecuados.* La tesis de mi co-tutor explica de forma detallada cómo trabajan estos plugins [6] y cuál es la relación que tienen los distintos aspectos del Pensamiento Computacional, además de demostrar cómo el desarrollo del Pensamiento Computacional ayuda a desarrollar habilidades que son compartidas con materias recogidas en los currículos del proceso Bolonia, por lo que aprender a programar puede ayudar a obtener mejores resultados en la enseñanza de asignaturas como Inglés, Matemásticas o Filosofía.

Dr. Scratch es un proyecto aún en desarrollo y ha tenido dos versiones hasta el momento. A continuación se muestran las dos versiones existentes de Dr. Scratch, tratando de explicar cuáles han sido las aportaciones de la versión anterior, para tener una visión del punto de partida con el que se contaba.

- La *versión Alpha* fué creada por el alumno Cristian Chusing, su principal aportación fue la creación de una página web inicial donde se pudiera subir un archivo con extensión .sb2 y mostrase un dashboard con toda la información obtenida con los pluguins de Hairball. La apariencia de dicha web es la mostrada en la *Figura 4.7*.

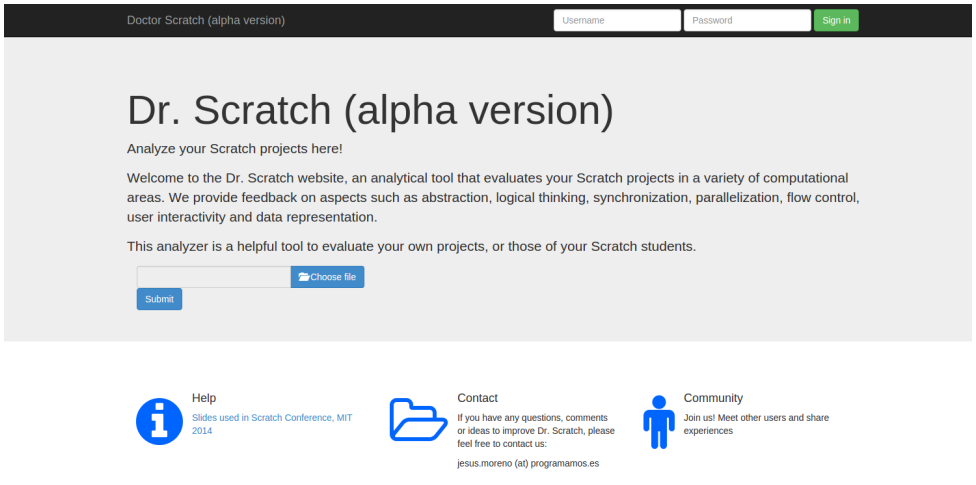

Figura 1.4: Apariencia de la versión Alpha de Dr. Scratch

[6] https://www.researchgate.net/publication/283118935_Automatic_Detection_of_Bad_Programming_Habits_in_Scratch_A_Preliminary_Study

- La *versión Beta* es la que he estado programando desde noviembre de 2014 con la ayuda de mi compañera Eva Hu Garres desde febrero de 2015. Ha sido realizado durante el periodo en el que fuimos contratadas a media jornada en la Universidad Rey Juan Carlos como técnicos de apoyo y difusión. Todo este proyecto se ha podido llevar a cabo gracias a la financiación otorgada por el FECyT (Fundación Española para la Ciencia y la Tecnología), además, de ser galardonado con un Google RISE Award en 2015 y del apoyo de la asociación sin ánimo de lugro Programamos. Esta versión es la que se va a detallar en este documento.

Al ser una plataforma en producción, se han realizado distintas pruebas en colegios, para comprobar el uso que realizaban los alumnos de ella, permitiéndonos observar cuáles eran las necesidades que demandaban. A lo largo de esta memoria se irán comentando los cambios introducidos en la herramienta y las razones que los han motivado.

1.4. Estructura de este documento

Intentando facilitar la lectura de esta memoria, se procede a explicar su estructura, detallando qué contenido tendrá cada capítulo:

1. **Introducción.** Explicación del marco social en el que se enmarca este proyecto, así como el marco de referencia en el que se ha desarrollado. Se muestra cuál ha sido la motivación para su realización y las fases iniciales del mismo.

2. **Objetivos.** Aquí se muestra cuáles han sido las metas a obtener para poder dar respuesta a la necesidad observada. Además, se indica cuáles han sido las funcionalidades a implementar. Éstas últimas, se mostrarán también, en una línea cronológica en las que se dividirá el tiempo en corto plazo, medio y largo, mediante un planteamiento temporal.

3. **Estado del arte.** En este capítulo, se hablará de las distintas tecnologías utilizadas para poder llevar a cabo la consecución del proyecto y se dará una explicación breve de las mismas.

4. **Diseño e implementación.** Tras conocer las tecnologías, entraremos a ver cómo las hemos utilizado para diseñar nuestra plataforma. Se explicará detalladamente cuál es la arquitectura de Dr. Scratch y todas sus funcionalidades.

5. **Interfaz de usuario.** Repaso rápido de la web y cuáles han sido sus modificaciones. Se indicará cuáles han sido las razones para los cambios en la mismas, mostrando así la evolución de la web según las necesidades del usuario recalcando las diferencias entre el prototipo inicial y el que está ahora mismo en producción.

6. **Resultados.** Al ser un proyecto en producción, conviene observar cuáles han sido los resultados obtenidos: número de proyectos analizados, número de organizaciones registradas e impacto en general.

7. **Conclusiones.** Capítulo en el que se realiza una reflexión general de los resultados obtenidos y cuál ha sido la consecución de los objetivos marcados inicialmente. Se indicará lo aprendido durante la formación en la Universidad Rey Juan Carlos y que ha sido aplicado en la realización de este Proyecto Fin de Carrera. Así mismo, también se presentarán cuáles son los conocimientos adquiridos durante la realización del proyecto Dr. Scratch y cuáles serían los siguientes pasos que se podrían dar en el mismo. Y para terminar, expresar mi valoración personal de todo lo que me ha aportado Dr. Scratch.

Capítulo 2

Objetivos

El principal **objetivo** de todo proyecto, es cubrir la necesidad que ha sido observada y que ha motivado la realización del mismo. Pero para poder llegar a dicho resultado es necesario detallar concretamente y sin ambigüedad qué es lo que queremos conseguir en cada etapa. En este capítulo se describen dichos objetivos iniciales, con la intención de poder reflexionar más tarde observando los resultados, cuáles han sido alcanzados y estudiar cuáles han sido los problemas o motivos que han repercutido en que no se logre alguno de ellos. Además, se indica el planteamiento temporal que se eligió para lograr alcanzar su consecución.

2.1. Objetivo general

El **objetivo** de este proyecto es el diseño y desarrollo de una herramienta web donde poder analizar proyectos programados en el lenguaje Scratch. El análisis nos reportará información de aspectos relacionados con el *Pensamiento Computacional* que debemos mejorar.

Para entender dicho objetivo, es necesario comprender qué es el pensamiento computacional y qué aspectos son los que queremos cubrir y cómo.

Cuando una persona empieza en el mundo de la programación, necesita desarrollar ciertas habilidades y cambiar en cierta medida el modo de plantear los problemas a los que se enfrenta. Necesita comenzar a pensar como lo haría un científico, lo que conlleva abordar los problemas de un modo diferente. Para ello, necesita desarrollar unas habilidades específicas que pueden ser muy útiles en otras disciplinas. Estas habilidades son las que se deben trabajar para conseguir un desarrollo del pensamiento computacional.

Jeannette Wing[1] en su artículo *Computational thinking*[2] indicó que el pensamiento computacional implica resolver problemas, diseñar sistemas y comprender el comportamiento humano, haciendo uso de los conceptos fundamentales de la informática.

La finalidad de Dr. Scratch es analizar el proyecto realizado en Scratch y evaluar dichos conceptos claves de las ciencias computacionales, en concreto, se centra en 7 aspectos:

1. **Paralelismo.** Consiste en poder resolver varios problemas de manera simúltanea en hilos diferentes.

2. **Pensamiento lógico.** Nos permite reconocer el problema que queremos solucionar, buscar una solución que podamos programar y analizar las salidas que obtengamos para formarnos conclusiones.

3. **Control de flujo.** Es la capacidad de poder coordinar el orden en que se ejecutan las instrucciones que tenemos en el programa.

4. **Interactividad con el usuario.** Habilidad que permite crear proyectos interactivos, que permitan que el usuario que ejecuta el programa, pueda realizar acciones que provoquen nuevas situaciones en el proyecto.

5. **Representación de la información.** Capacidad de un programador de detectar los datos que el programa necesita, para poder ejecutarse correctamente.

6. **Abstracción.** Concepto que ayuda a dividir un problema en partes más pequeñas que serán más fáciles de comprender, programar y depurar.

7. **Sincronización.** Es la comunicación entre las distintas partes del programa que permite la planificación de la ejecución de las instrucciones, en un orden determinado que ha sido elegido con anterioridad.

Todos estos conceptos han de ser evaluados de forma que el usuario al final obtenga una **puntuación total que califique el nivel de desarrollo del pensamiento computacional del programador de proyectos en Scratch**. La evaluación se realiza conforme a los bloques de Scratch utilizados y es parte del trabajo de Jesús Moreno en su tesis.

[1] `http://www.cs.cmu.edu/~wing/`

[2] `https://www.cs.cmu.edu/~CompThink/papers/Wing06.pdf`

Pero Dr. Scratch va más allá. Incluye además información sobre malas **prácticas realizadas en programación** observadas en sus años de educación de mi tutor y co-tutor y que todo programador novel realiza en sus primeros proyectos -y últimos si no aprende a corregirlos-.

De esta forma, Dr. Scratch trata de dar una puntuación basada en el nivel computacional del programador, pero además sirve de apoyo para aprender a programar correctamente. Esto la convierte en una herramienta muy útil, tanto para docentes como estudiantes.

2.2. Objetivos específicos

Una vez detallado el objetivo general del proyecto, necesitamos entender en qué momento de su desarrollo nos encontramos y cuáles son los objetivos que queremos alcanzar al final de este Proyecto Final de Carrera.

Tal como se ha comentado, ya existía una versión Alpha de la web, en la que se tenía un código, que trataba de alcanzar el objetivo e implementación básica de la web. Dicho código, mostraba una web simple e intentaba analizar el archivo .sb2 de Scratch para extraer conclusiones mediante el plugin Hairball y mostrarlas en un dashboard.

Al comienzo de este proyecto, dicho código no conseguía realizar correctamente el análisis del proyecto de Scratch, por lo que partiendo de esta base, los objetivos iniciales fueron los siguientes:

1. **Corrección de la implementación básica.** El primer objetivo y más importante fue conseguir tener en funcionamiento el prototipo que ofrecía la funcionalidad básica.

2. **Generación de un log.** Se quería tener información de todos los proyectos analizados en la web: quién lo había analizado -si estaba registrado-, nombre del proyecto, hora y fecha.

3. **Plataforma en producción.** La idea principal, era que una vez el prototipo funcionase, tenerlo en producción lo antes posible para ser utilizado y poder tener feedback de cuáles eran las necesidades de los usuarios.

4. **Cambio de apariencia de la web.** La web funcionaba, pero no era demasiado atractiva y se buscaba tener el máximo número de usuarios posibles, por lo que era necesario realizar un cambio. Teníamos además, que buscar algún logotipo que mostrase la esencia de Dr. Scratch.

5. **Implementación del mecanismo de traducción de la plataforma.** Se busca llegar con Dr. Scratch al mayor número de personas y para ello el idioma no debe ser un impedimento. Por ello un objetivo inmediato fue la configuración del servidor para que fuera capaz de traducir la web al idioma del navegador que hiciera la petición.

Dr. Scratch ha sido un proyecto que ha obtenido financiación por parte del **FECyT** (Fundación Española para la Ciencia y la Tecnología) y por tanto, hay una serie de objetivos iniciales que fueron fijados en la memoria de solicitud de la ayuda. Tenía objetivos referentes tanto a la plataforma web como a otros subobjetivos, tales como la realización de talleres para garantizar la difusión de la plataforma y la creación de materiales docentes, entre otros. En este documento únicamente nos centraremos en los objetivos específicos relacionados con la plataforma web:

- Llegar a conseguir más de 5.000 proyectos analizados.
- Número de alumnos registrados ($>$1.500)
- Número de docentes registrados ($>$120)
- Número de visitantes únicos diarios de la web $>$ 25.000
- Número de descargas de la aplicación móvil ($>$ 150)
- Número de horas del servidor caído en los últimos 6 meses de proyecto ($<$ 100h)
- Número de lenguas de la plataforma web ($\geq$5)

Nada más tener el prototipo en producción, nos llegaron las primeras peticiones de los usuarios, y por tanto, más objetivos para nosotros y algunos más que surgieron con el avance del proyecto:

- Barra de progreso del análisis del proyecto.
- Analizar proyectos de las dos versiones de Scratch (1.4 y 2.0), únicamente se podía analizar proyectos de la versión 2.0, si intentábamos analizar proyectos de la 1.4 la plataforma mostraba un error.
- Análisis mediante url del proyecto de la versión online de Scratch.

- Cuentas de usuarios.
- Implementación en el servidor para mostrar tres páginas diferentes, conforme el nivel del programador fuera: bajo, medio o alto.
- Diseño de tres páginas distintas para cada nivel computacional.
- Debido al gran número de proyectos que se analizaban diariamente teníamos numerosos problemas con las máquinas de la universidad, por lo que decidimos realizar una migración a la nube.
- Utilización de algún mecanismo de analisis web para tener estadísticas de los accesos a la plataforma.
- Páginas con información de cómo poder pasar de un nivel a otro utilizando distintos bloques de Scratch que demuestran un nivel de pensamiento computacional mayor.
- Análisis para organizaciones, permitiéndoles analizar varios proyectos de Scratch simultáneamente a través de un fichero .csv y le devolviera toda la información del análisis.
- Cuentas de organizaciones.
- Páginas de estadísticas que permitan obtener información acerca de la evolución de los usuarios y poder realizar un seguimiento.
- Extensión para los navegadores que permitiera analizar un proyecto desde la propia página web de Scratch. De forma que únicamente clicando en un botón, la extensión obtuviera la url del proyecto subido y compartido en Scratch mostrado en la página actual y abriese una nueva ventana con su análisis en Dr. Scratch.
- Además de la traducción automática detectando el idioma mediante la configuración del navegador, tener un botón que permita al usuario modificar el idioma.
- Página de agradecimiento a los colaboradores: personas que han traducido los textos de la web a distintos idiomas, colegios que nos han permitido realizar talleres en sus aulas...
- Un foro donde los usuarios pudieran expresas sus opiniones y sugerencias para poder obtener feedback.

- Dashboard para docentes, que mostraran información al maestro sobre sus alumnos.

En última instancia, se tiene en mente que la web incluya mecanismos de gamificación para conseguir motivar a los usuarios a seguir aprendiendo utilizando la herramienta, y además, al más estilo del lenguaje Scratch, tener una red social, que permitiera a los programadores poder interactuar y dinamizar la plataforma.

2.3. Planificación temporal

Al ser un proyecto tan amplio y ser desarrollado en equipo por varias personas, quiero dejar lo más detallado posible cuál ha sido el planteamiento temporal y la participación de todos los integrantes del equipo Dr. Scratch y aportaciones realizadas por Cristian Chusing en la primera versión Alpha. En este punto, únicamente desglosaré la planificación temporal, a la que en próximas secciones iré incluyendo qué miembros han participado en la consecución de cada funcionalidad y objetivo.

Este proyecto ha sido muy bien guiado y gestionado por mi tutor y co-tutor mediante **Trello** -herramienta colaborativa para organizar proyectos-. Y fueron marcados incialmente, cuáles eran los objetivos a corto, medio y largo plazo. Según el proyecto fue evolucionando, dichas metas podían ser modificadas fácilmente gracias a la herramienta y jamás se perdía de vista cuáles eran las tareas más inmediatas.

- **FASE I.** Esta etapa del proyecto es la realizada desde noviembre de 2014, fecha de comienzo, y febrero de 2015, cuando se incorporó al proyecto mi compañera Eva Hu. Durante este periodo los objetivos a alcanzar fueron los siguientes:

    - Comprender y depurar el código realizado por Cristian Chusing en la versión Alpha de Dr. Scratch.
    - Actualización de la versión 1.4 de Django a la 1.7.
    - Generación del archivo log explicado con anterioridad.
    - Puesta en producción del prototipo.
    - Cambio de apariencia de la web.
    - Implementación del mecanismo de traducción de la plataforma.

En la *Figura 4.8* se muestra dicha fase de una forma esquemática y sencilla de comprender.

Figura 2.1: Fase I del proyecto Dr. Scratch

- **FASE II.** Esta etapa del proyecto es la realizada desde febrero a julio de 2015.

  En esta fase del proyecto los objetivos a alcanzar fueron los siguientes:

  - Introducir una barra de progreso del análisis del proyecto.
  - Corrección de errores en formularios.
  - Análisis mediante url del proyecto de la versión online de Scratch.
  - Migración de SQLite a MySQL.
  - Creación de un archivo en el que se guardaran todos los proyectos en los que se producía algún error al ser analizados.
  - Personalización de las páginas de error 404 y 500.
  - Analizar proyectos de las dos versiones de Scratch (1.4 y 2.0).
  - Implementación en el servidor para mostrar tres páginas diferentes, conforme el nivel del programador fuera: bajo, medio o alto.
  - Diseño de tres páginas distintas para cada nivel computacional.
  - Introducción explicaciones de qué es una url y cómo descargar un proyecto en Scratch.
  - Migración a la nube en lugar de utilizar las máquinas de la universidad, las cuales nos dieron muchos problemas.

- Uso de Google Analytics para tener estadísticas de acceso a la plataforma.
- Páginas explicativas de cómo subir de nivel utilizando bloques de un mayor nivel computacional.
- Nuevo diseño de los dashboards.
- Cuentas de organizaciones.
- Botón de "Ayuda" para saber cómo navegar por el dashboard.
- Análisis multiproyecto con un archivo .csv para organizaciones.
- Traducción de la información mostrada en el archivo de organizaciones.
- Páginas de estadísticas de los proyectos analizados.
- Generación de un archivo .csv que muestra la información obtenida con el analáisis realizado en Dr. Scratch y es el que se le devuelve a la organización.

En la *Figura 4.9* podemos ver otro esquema que escenifica esta fase.

Figura 2.2: Fase II del proyecto Dr. Scratch

- **FASE III.** Abarca el periodo entre septiembre de 2015 y 1 de febrero de 2016, fecha en la que empecé a escribir esta memoria.

  Trás la Scratch Conference en Ámsterdam a la que acudió todo el equipo de Dr. Scratch y en la que realicé una presentación de la plataforma, pudimos obtener algunas conclusiones gracias a la realimentación obtenida. Debido a ellas, marcamos los siguientes objetivos:

  - En el análisis por URL únicamente descargar el .JSON para que se realice más rápidamente.

- Traducción de los bloques pintados en las páginas para aprender cómo subir de nivel de pensamiento computacional.
- Registro de usuarios.
- Botón para traducir la página según la preferencia del usuario y no del navegador.
- Extensión que permita analizar proyectos sin necesidad de abrir en el navegador la propia página de URL, sino que en la página del proyecto de Scratch nos facilite un botón que ya muestre directamente el análisis realizado.
- Traducción de los textos que aparecen en la plataforma a más idiomas: catalán, gallego y portugués.
- Formulario de registro para usuarios y organizaciones.

En la *Figura 4.10* podemos observar estos objetivos detallados en el tiempo.

Figura 2.3: Fase III del proyecto Dr. Scratch

Actualmente, el proyecto continúa desarrollandose, con la ayuda de la Asociación sin ánimo de lucro **Programamos** en la que mi co-tutor, Jesús Moreno, es co-fundador junto con José Ignacio Huertas.

Capítulo 3

Estado del arte

En este capítulo se detallarán de forma resumida las **arquitecturas y tecnologías** utilizadas en la realización de este proyecto. Únicamente se realizará una presentación de las mismas, ya que en el próximo capítulo será donde se detalle cómo han sido utilizadas para el desarrollo e implementación de la plataforma.

3.1. Aplicaciones cliente-servidor

El paradigma **Cliente-Servidor** [3] es utilizado en diversos contextos. Centrándonos en el patrón arquitectural para software distribuido cabe indicar que es un modelo en el que existen dos tipos de elementos -cliente y servidor- y un modelo de interacción basado en un diálogo -petición y respuesta-.

El objetivo de esta arquitectura es **proporcionar el soporte escalable** para el desarrollo de aplicaciones distribuidas.

En cuanto a sus **elementos**:

- **El cliente.** Es la parte de la aplicación con la que interacciona el usuario pero no es compartida entre los mismos. No tiene restricciones especiales referentes a fiabilidad, ya que si falla un cliente, el resto del sistema no se ve afectado. Pero sí las tiene en términos de ergonomía debido a que debe adaptarse a la interacción con el usuario. Este último aspecto en concreto, será tratado con profundidad en el siguiente capítulo, ya que ha sido la mayor preocupación a la hora de diseñar la plataforma.

- **El servidor.** Es el componente que presta servicios al cliente, compartiendo sus recursos con todos ellos. Tiene una gran restricción y es que si el servidor falla, los clientes no pueden continuar. Además de presentar problemas de escalabilidad y debe ser seguro para no comprometer la seguridad de los clientes ni de los datos.

En la *Figura 3.1* podemos observar sus componentes y el intercambio de mensajes. El protocolo de intercambio de mensajes será explicado más adelante.

Figura 3.1: Esquema del paradigma Cliente-Servidor

También es importante entender que la finalidad de una aplicación distribuida es la de ofrecer funcionalidades al usuario y que su **estructura** es siempre la que se muestra en la *Figura 3.2*.

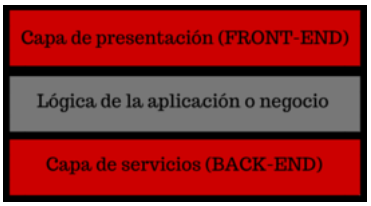

Figura 3.2: Capas presentes en toda aplicación distribuida

1. **Capa de presentación.** También conocida como front-end, suele residir en el cliente y es con la cual el usuario interactua. Se pueden utilizar distintas tecnologías como: HTML, CSS, javascript...

2. **Lógica de la aplicación o negocio.** Puede residir en cualquiera de los componentes -cliente y servidor- incluso en ambos. Realiza acciones necesarias para desarrollar la actividad de negocio, tales como: cobro de producto, transacciones...

3. **Capa de servicios.** Denominada como back-end, es común que resida en el servidor y realiza las actividades típicas de gestión de ficheros, acceso a datos en la base de datos, procesamiento de la información...

Podemos encontrarnos distintos **tipos de servidores**:

1) Dependiendo de si la aplicación requiere que el servidor recuerde la historia pasada del cliente, se diferencian dos tipos:

a) Servidor con estados: recuerda información entre las peticiones.

b) Servidor sin estados: no recuerda información del cliente entre peticiones.

2) Atendiendo al modo de procesamiento de las peticiones, encontramos dos servidores diferentes:

a) Servidor concurrente: se atienden las peticiones de forma paralela, gracias a la implementación multihilo.

b) Servidor iterativo: se atienden las peticiones de forma secuencial, existiendo una cola de peticiones.

Hypertext Transfer Protocol

El intercambio de mensajes se realiza mediante el protocolo HTTP, el cual es un protocolo sin estado y ha tenido varias versiones. De estas versiones la más utilizada es la versión 1.1 HTTP que especifica la sintaxis y semántica que utilizan cliente y servidor para comunicarse.

La estructura de los mensajes de petición y respuesta según el protocolo HTTP se muestran en la *Figura 3.3*. Son en texto plano lo cual es una ventaja a la hora de depurar pero hace los mensajes largos.

Figura 3.3: Mensajes de petición y respuesta del protrocolo HTTP

3.2. Back-end

Tal como hemos visto en la explicación de la arquitectura cliente-servidor el *back-end* se corresponde con la **capa de servicios o de acceso a datos**. Y en esta sección vamos a ver la parte teórica de las tecnologías utilizadas en este proyecto para el desarrollo de dicha capa:

1. El lenguaje de programación elegido: Python
2. El framework utilizado para la implementación del servidor: Django
3. Servidores HTTP utilizados: Apache y Node.js
4. La base de datos configurada: MySQL

3.2.1. Python

El **lenguaje Python** [1], cuyo nombre es en homenaje a la serie *Monty Python's Flying Circus*. Fue desarrollado por *Guido van Rossum* para tratar de superar las limitaciones del *lenguaje ABC*, que era comúnmente utilizado en el centro de investigación en el que trabajaba en Ámsterdam.

Desde el principio se buscó que fuera un *lenguaje divertido* a la hora de ser utilizado y es común utilizar variables que hacen referencia a sketchs de los Monty Python en lugar de variables tradicionales.

En un principio iba a publicarlo bajo una licencia open source propia, pero con la versión 1.6 se decidió por cambiar a **licencia GPL** (GNU General Public Licence). Así, es posible modificar el código fuente y desarrollar código derivado sin la necesidad de hacerlo open source.

Existen **tres versiones** principales, con 1.6, 2.7 y 3.4 como últimas actualizaciones. Es destacable indicar que las versiones 2 y 3 son incompatibles entre sí, por el gran número de *diferencias entre ellas* [4].

El desarrollo y promoción se realiza a través de la organización, sin ánimo de lucro, *Python Software Foundation*, destacando su **carácter open source**.

Las **características** principales de este lenguaje son las siguientes:

- Es un lenguaje de *alto nivel*, expresando el algoritmo de forma entendible para el programador.

- *Interpretado*, se ejecuta mediante un intérprete y así es más flexible que un lenguaje compilado.
- Su filosofía se centra en tratar de conseguir un *código lo más legible posible* para el ser humano. Los programas en Python parecen pseudocódigos.
- Es *orientado a objetos*, con sus clases, propiedades, métodos. . . . Todo son objetos.
- *Dinámicamente tipado*: puede indicarse el tipo en cualquier momento, no es necesario declararlos en un método. Esto reduce el ciclo editar-compilar-comprobar-depurar.
- *Fuertemente tipado*: una vez declarada una variable con un tipo, esta no puede realizar una violación de tipo de datos, sino que deberá ser convertida previamente.
- *Case sensitive*: diferencia entre mayúsculas y minúsculas.

Ha sido el lenguaje elegido, debido principalmente a dos razones, es el lenguaje que me ha resultado más atractivo en mis años de aprendizaje en la universidad, y en segundo lugar, por su amplia utilización mundial. Las estadísticas mostradas por CodeEval[1] para 2015 de los lenguajes más utilizados pueden observarse en la *Figura 3.4*. En 2014, el dato para Python era de 30,3 %, por lo que sigue aumentando el número de programadores que se decantan por este lenguaje.

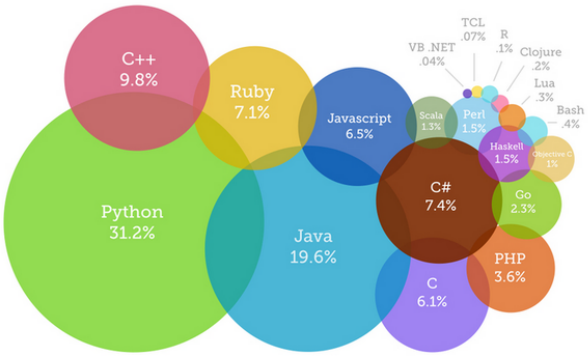

Figura 3.4: Lenguajes de programación más utilizados en 2015

Por último, destacar que Python es un **lenguaje multiplataforma** por lo que se puede utilizar en diversas plataformas. Y cuenta con varios *web frameworks*, entre los que destaca Django, que es la tecnología que trataré a continuación.

[1] `https://www.codeeval.com/`

3.2.2. Django

Django es un framework que facilita al programador la creación de sitios webs complejos. Entre sus principales **características**, destacar:

- Es de *código abierto*.
- Todo está escrito en *Python*.
- Sigue el concepto *DRY* -Don't Repeat Yourself- intentando no duplicar código.
- El programador no tiene que preocuparse de realizar las tareas repetitivas que siempre hay que realizar al crear un servidor web: abrir sockets, cerrar sockets, configuración de protocolos...
- Puede ser utilizado con varias bases de datos, tales como MySQL, SQLite o PostgreSQL.
- Su filosofía de desarrollo es MVC -Modelo Vista Controlador-, pero no en sentido estricto, sino que "hacen lo que les parece correcto". De tal forma, podemos encontrar nombres distintos a los comúnmente utilizados en un framework MVC puro. Se podría decir que es un modelo MTV -Modelo Template Vista-.

Para explicar esta última característica y cómo funciona el modelo de Django en concreto, he creado la *Figura 3.5*[2].

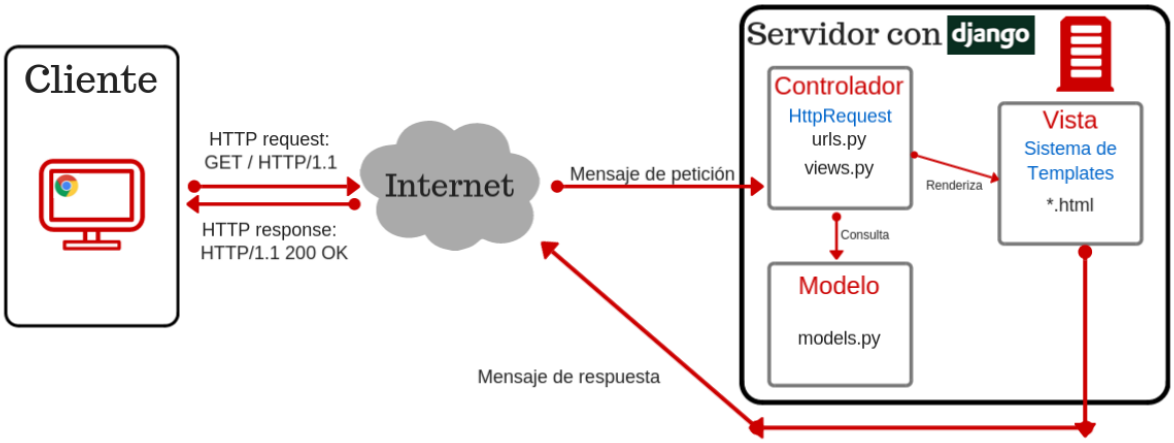

Figura 3.5: MVC adaptado de Django

[2]Para la creación de esta figura me he apoyado en la información encontrada en `http://es.slideshare.net/alatar/django-el-framework-web-definitivo-1362169`

Al crear un proyecto nuevo, tendremos por defecto el árbol de directorios mostrado en la *Figura 3.6*, donde se pueden observar algunos de los ficheros mencionados al describir el modelo seguido por Django.

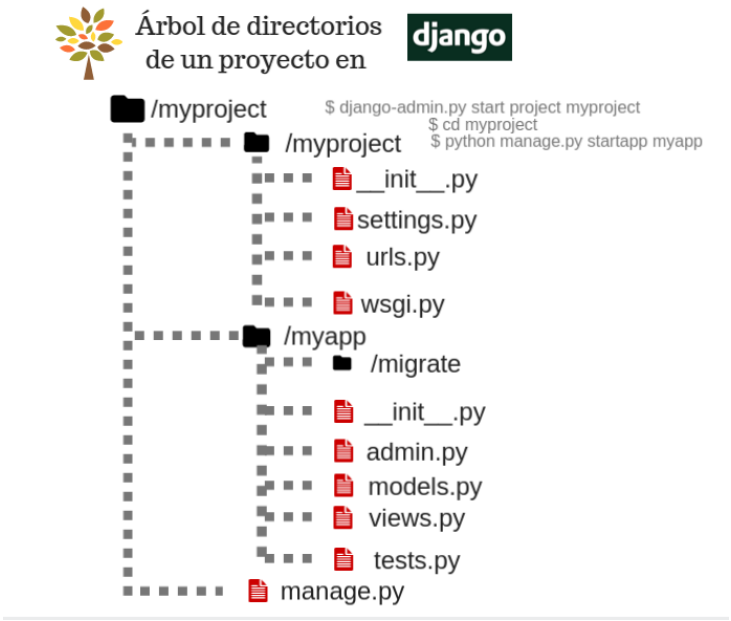

Figura 3.6: Árbol de directorios de un proyecto en Django

Django tiene una gran **comunidad** de usuarios que defienden al igual que Python las buenas *prácticas de programación y que el código sea legible* [2].

3.2.3. MySQL

Todo servidor web necesita un *servidor de base de datos* que administre, almacene y gestione los datos, para asegurar el acceso seguro a ellos.

MySQL [5] es uno de los sistemas de administración de bases de datos más utilizados en la actualidad, se puede observar en la *Figura 3.7*[3] cuál es el ranking que ocupa relacionado con otras bases de datos muy utilizadas en el mercado, donde se puede observar que es una de las tres con más usuarios.

[3]Información obtenida en `http://db-engines.com/en/ranking_trend`

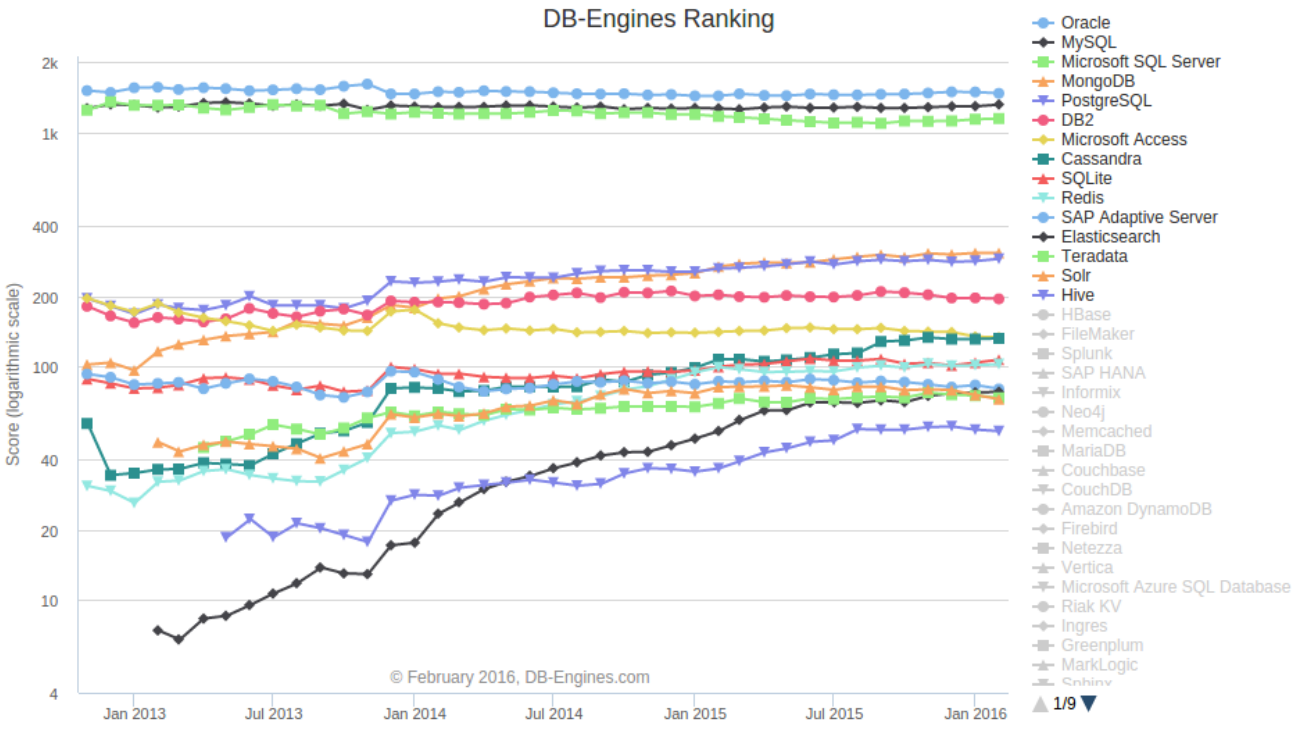

Figura 3.7: Ranking de las bases de datos más utilizadas actualmente

Las principales **características** de MySQL son las siguientes:

- Utiliza **tablas** para guardar los datos.
- Tiene un mecanismo interno de almacenamiento el cual guarda los datos salvados en dispositivos físicos.
- Es **relacional**(BDR), por lo que permite establecer interconexiones o relaciones entre los datos ya guardados en tablas, siguiendo el *modelo relacional*.
- Es **multiplataforma**.
- Utiliza multihilos mediante hilos del kernel.
- Los clientes se conectan al servidor MySQL usando sockets TCP/IP.
- Tiene bases de datos de hasta 50 millones de registros.

Destacar que tiene **licenciamiento dual**: se puede utilizar bajo la GNU GPL, pero si se quiere incorporar en productos privativos es necesario comprar otra licencia.

3.2.4. Servidor HTTP Apache

El **servidor HTTP Apache** es un servidor web HTTP de *software libre*. El servidor se desarrolla y mantiene bajo la *Apache Software Foundation* dentro del proyecto HTTP Server *(httpd)*.

La arquitectura del servidor Apache es **altamente modular**, lo cual permite añadir más funcionalidades de modo sencillo y ser extendido fácilmente. Esta es una de las características que le hacen ser muy utilizado a la hora de poner una web en producción, ya que se pueden incluir módulos de frameworks. Como es el caso del módulo *mod_wsgi de Django*.

Algo históricamente notable es la licencia de software bajo la cual es distribuido. La **Licencia Apache** es una licencia de software libre y permite al usuario la libertad de uso para cualquier propósito, ya sea de distribución, para ser modificado y le permite además, distribuir versiones modificadas del software.

La **configuración** del mismo se realiza mediante el fichero apache2.conf o **httpd.conf** y para que los cambios en él sean agregados es necesario reiniciar el servidor.

3.2.5. Servidor HTTP Node.js

Node.js es un entorno de ejecución que permite trabajar sobre el intérprete de *JavaScript* en el lado del servidor. Utiliza el motor V8, desarrollado por Google para el navegador Chrome, compilando javascript en el lado del servidor a altas velocidades.

Señalar que es de código abierto y multiplataforma. Y la característica que lo hace destacar es el hecho de poseer un excelente *modelo de eventos*, ideal para la programación asíncrona.

3.3. Front-end

Recordar que el *front-end* es la **capa de presentación** que toda aplicación distribuida posee. Es la parte con la que el cliente interacciona. Y en las siguientes subsecciones se van a explicar algunas de las tecnologías más utilizadas actualmente en el desarrollo web y que se muestran en la *Figura 3.8*.

Figura 3.8: Tecnologías más utilizadas en la capa de presentación

3.3.1. HTML

HTML (HyperText Markup Language) no es un lenguaje de programación, sino un *lenguaje de marcado* para la creación de páginas webs. Es utilizado por todos los navegadores actuales.

Permite dar estructura e introducir contenido en las páginas webs a través de las distintas **etiquetas** que componen el lenguaje, utilizando únicamente texto. En la *Figura 3.9* se puede observar la estructura básica de un código HTML.

```
<!DOCTYPE html>
<html>
    <head>
        <!-- Información técnica para el navegador-->
    </head>
    <body>
        <!-- Contenido que aparecerá en la página web-->
        <p>Hello, World!</p>
        <img class="col-md-6" src="static/app/images/drScratch.png"/>
    </body>
</html>
```

Figura 3.9: Estructura básica de un código HTML

Para incluir un **elemento** en una página necesitamos indicar la etiqueta de inicio, su contenido -texto que aparecerá en la web- y la etiqueta de cierre, generalmente. Las etiquetas de inicio tienen **atributos** en los que se especifica mediante texto plano la ubicación de los contenidos(imágenes, vídeos...) que queremos introducir en la página y es el navegador quien se encarga de interpretarlo y mostrar la página web completa.

3.3.2. CSS

CSS (cascading style sheets) es utilizado para dar estilo a un lenguaje estructurado, como puede ser HTML. Se aconseja que el código CSS se incluya en otro documento separado del HTML y que éste sea incluido mediante la etiqueta adecuada(¡link¿), tratando de *no mezclar lenguajes diferentes en un mismo texto* para facilitar su legibilidad.

Al seguir la norma de estilo anterior, el modo de indicar a un elemento de HTML que debe seguir una norma de estilo concreta se realiza incluyendo en la etiqueta de dicho elemento alguno de los dos atributos de nombre: id o class. Estos dos atributos tendrán un valor determinado que será el especificado en una sección del código CSS que tendrá unas reglas propias.

El mayor potencial de CSS es su **sintaxis simple** pero hay que tener mucho cuidado con las jerarquías para incluir el estilo que deseamos a los contenidos deseados.

3.3.3. Bootstrap

Bootstrap es un framework que permite al usuario despreocuparse en gran medida del diseño y apariencia al realizar una aplicación web. Entre sus principales aportaciones están las numerosas **plantillas** que permiten tener implementado el diseño de una página web completa en pocas horas.

Está *basado en HTML y CSS* separando por completo dichos lenguajes en documentos diferentes lo que permite que una plantilla pueda ser modificada y ampliada fácilmente por el usuario para adaptarlo al diseño deseado de un modo intuitivo.

Las plantillas generadas, normalmente incluyen también ficheros con *JavaScript* para aportar a la página un diseño más atractivo para el usuario, en la siguiente subsección se explicará qué mejoras introduce este lenguaje a una página creada únicamente con HTML y CSS.

3.3.4. JavaScript

JavaScript es un lenguaje de programación interpretado utilizado de forma extendida para poder *dinamizar el lado del cliente* incluyendo así mejoras en la interfaz de usuario y aumentar su seguridad.

Aun que su principal uso es en el lado del cliente, también se utiliza con el apoyo de otras tecnologías para enviar y recibir información al servidor.

3.3.5. AJAX

AJAX (Asynchronous JavaScript And XML) es una tecnología que permite enviar y recibir información desde el servidor al cliente, permitiendo diseñar páginas webs más interactivas.

Este envío de información se realiza gracias a una *comunicación asíncrona* con el servidor siendo posible realizar cambios en la página siendo recargada automáticamente.

3.4. Plataforma en la nube: Azure

Siempre que se persigue trasladar una aplicación distribuida a producción es necesario una máquina o máquinas en las que implementar todas las tecnologías que han sido mencionadas anteriormente. **Microsoft Azure** es una plataforma alojada en los Data Centers de Microsoft que permite entre otros servicios, la posibilidad de tener máquinas virtuales con una gran selección de sistemas operativos.

Por ello, es una herramienta muy potente en la que alojar nuestra aplicación asegurando *escalabilidad, seguridad y un 24x7 fiable*.

3.5. Analítica web: Google Analytics

El objetivo principal de una web en producción es que sea utilizada por el mayor número de usuarios posible y para poder realizar un seguimiento y medición de dicho objetivo **Google Analytics** nos permite tener numerosas *estadísticas* y herramientas que nos reportan dicha información.

Su utilización es muy simple, al crear una cuenta, *se genera un código en JavaScript que debe ser incluido en las distintas páginas de la web*.

Además, Google Analytics incluye numerosa información en texto y vídeos de cómo leer la información dada y de cómo optimizar aspectos tan importantes en las webs como es la segmentación del tráfico o publicidad online.

Capítulo 4

Diseño e implementación

En este capítulo se va a detallar *cómo se han implementado las distintas tecnologías* explicadas en el capítulo anterior para lograr el desarrollo de la plataforma de Dr. Scratch. También se irá explicando cuál ha sido la *evolución en la implementación de las funcionalidades debido a las necesidades observadas en los test de usabilidad* realizados en dos colegios de la Comunidad de Madrid con alumnos de 5º y 6º de primaria: CEIP Lope de Vega y CEIP Gonzalo Fernández de Córdoba.

4.1. Arquitectura general de Dr. Scratch

Como toda aplicación Cliente-Servidor, Dr. Scratch tiene las 3 capas de desarrollo: back-end, lógica de la aplicación y front-end.

Pero al ser Dr. Scratch un analizador de archivos -en este caso, el JSON del proyecto de Scratch- la lógica o negocio de la aplicación está clara. Por ello se va a prestar mayor atención a la capa de servicio y de presentación.

4.1.1. Capa de servicio: back-end

En la *Figura 4.1* se puede observar la arquitectura de Dr. Scratch y la conexión entre las distintas tecnologías utilizadas para la implementación de la capa de servicio.

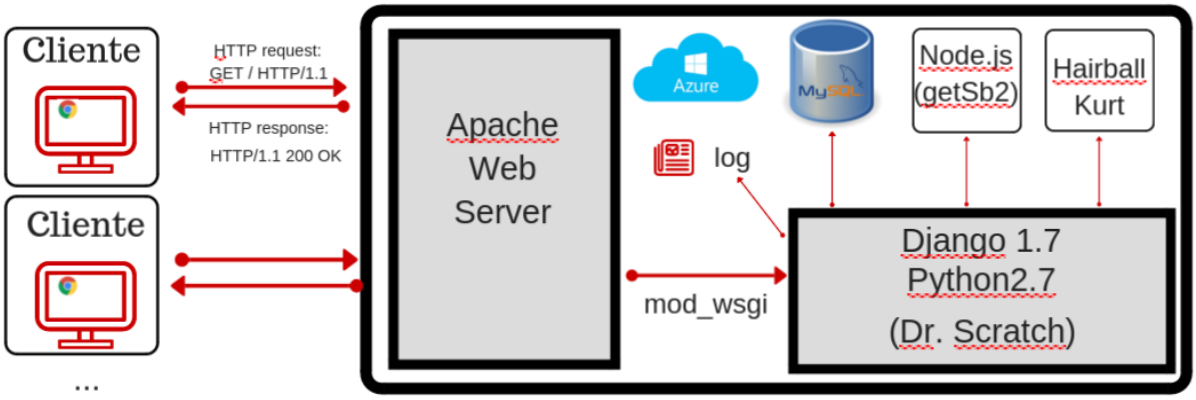

Figura 4.1: Arquitectura general de Dr. Scratch

Azure

Todo lo que se va a proceder a explicar a continuación ha sido instalado y programado en una máquina virtual en la nube mediante la plataforma Microsoft Azure con un sistema operativo Ubuntu 14.04.2 LTS.

Esto ha sido posible porque Microsoft nos ha ofrecido una suscripción de 150$ al mes de forma gratuita. Por ello, al principio pensamos que era mejor utilizar la opción website de la plataforma ya que nos indicaron que esa opción consumía menor crédito al viajar menos tráfico. Publiqué un website de Dr. Scratch con su base de datos MySQL enlazada a través de Visual Studio pero el problema que se encontró fue que en los websites de Azure no hay posibilidad de instalar otras aplicaciones y por ello no teníamos la posibilidad de instalar Hairball. Intentamos implementarlo ya que Python ofrece la posibilidad de ser ejecutado indicando el path completo en el que se encuentran los ficheros importando todos los módulos que había en los ficheros de Hairball y sus librerías de forma absoluta, pero había librerías dinámicas que presentaban más problemas. En este momento, probamos a crear una máquina virtual instalando absolutamente todo lo que necesitábamos dentro de ella y controlar el consumo del crédito. La sorpresa fue que todos los meses, aún teniendo miles de proyectos analizados, el crédito no se consumía en más de un 70 %.

Apache

Actualmente se encuentra instalada la versión 2.4.10 ya que es la compatible con la versión de Ubuntu de nuestra máquina. La configuración del fichero httpd.conf me llevó algún tiempo entenderla, ya que era necesario además incluir el módulo mod_wsgi que permite

enlazar Azure con Django. La tarea realizada por Apache nos resuelve numerosos problemas y su funcionamiento es el siguiente: cuando los clientes realizan peticiones HTTP a nuestro dominio a través de la red, estas son escuchadas por el servidor de HTTP implementado bajo la licencia Apache, la cual permite utilizar Dr. Scratch de forma simúltanea por numerosos usuarios teniendo estos la percepción de ser los únicos usándolo.

Django

La versión actualmente instalada de Django es la 1.7 ya que la 1.8 está aún en desarrollo. Como se observa, Apache se encarga de hacer llegar la petición a Django, donde tal como se puede ver en la *Figura 4.2* se pregunta en el fichero urls.py si dicho recurso es servido por nuestro servidor. El fichero urls.py es básicamente una lista en la que mediante expresiones regulares se indica qué recursos son servidos por nuestra aplicación.

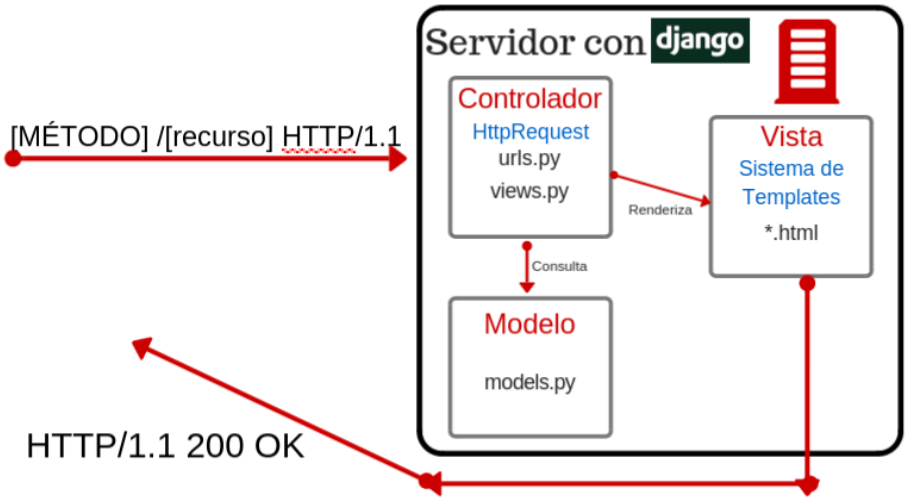

Figura 4.2: Modelo de Dr. Scratch

Dr. Scratch se ha implementado de tal forma que si se realiza una petición a nuestro dominio con un recurso que no es servido redirija al usuario a la página principal, así si el usuario, que puede ser un niño pequeño, se equivoca, pueda seguir navegando sin mayor esfuerzo o ver alguna página de error. Internamente, la lógica de este funcionamiento es el mostrado en el diagrama de la *Figura 4.3*.

Figura 4.3: Funcionamiento de la petición de recursos a Dr. Scratch: urls.py

En el momento que urls.py nos dirige a una función -recordar que en Python no existen los procedimientos- las cuales se encuentran según el modelo de Django en el fichero views.py, se debe hacer distinción según si el método de la petición es: GET, POST u otro método. Así se ejecutará un código concreto según la función y el método. Algunos métodos únicamente servirán el recurso solicitado si el método que acompaña a la petición es GET y en otro caso redirigirá a la página principal de Dr. Scratch para que al igual que en el caso anterior el flujo de la aplicación sea lo más dinámica posible. En la *Figura 4.4* se observa el funcionamiento interno.

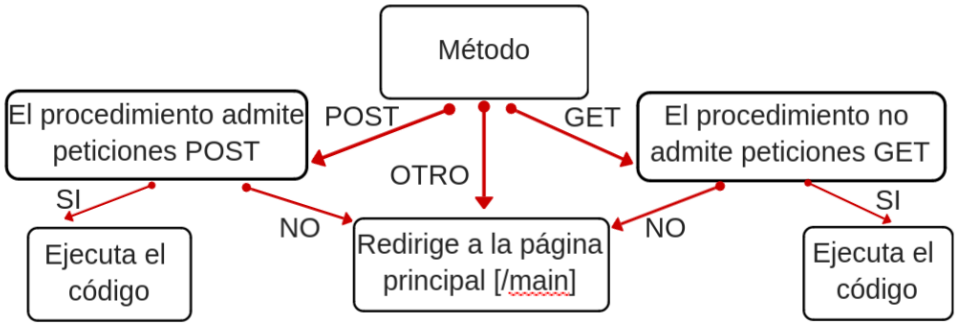

Figura 4.4: Distinción según el método de la petición: views.py

Al ser Dr. Scratch una plataforma con numerosas funcionalidades, fue necesario apoyar al fichero views.py con otra serie de ficheros en los que se implementaban funciones auxiliares. Así se permite observar fácilmente el flujo de la plataforma y aumentar su modularidad. Los otros ficheros que apoyan a views.py se pueden ver en la *Figura 4.5*.

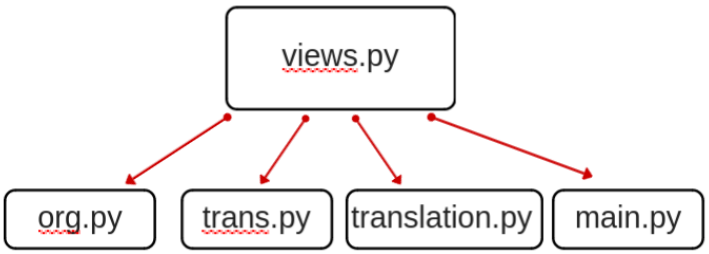

Figura 4.5: Modularización del fichero views.py

Para realizar el código de las funciones implementadas en los ficheros anteriores hay que recordar que Django está completamente programado en Python y que este es un lenguaje orientado a objetos, por ello, todo lo que encontraremos en nuestro código son objetos. La declaración de los objetos que conforman nuestra aplicación se encuentra en el fichero models.py del modelo de Django. La declaración de uno de los objetos más utilizados por Dr. Scratch se muestra en la *Figura 4.6* donde se pueden observar sus campos. Las distintas clases declaradas en este fichero serán instanciadas en las funciones para implementar las diferentes funcionalidades de la plataforma.

```
import datetime
from django.db import models
from django.contrib.auth.models import User

# Models of drScratch


class File(models.Model):
    filename = models.CharField(max_length=100)
    organization = models.CharField(max_length=100, default='drscratch')
    coder = models.CharField(max_length=100, default='drscratch')
    method = models.CharField(max_length=100)
    time = models.DateField(auto_now=False)
    language = models.TextField(default="en")
    score = models.IntegerField()
    abstraction = models.IntegerField()
    parallelization = models.IntegerField()
    logic = models.IntegerField()
    synchronization = models.IntegerField()
    flowControl = models.IntegerField()
    userInteractivity = models.IntegerField()
    dataRepresentation = models.IntegerField()
    spriteNaming = models.IntegerField()
    initialization = models.IntegerField()
    deadCode = models.IntegerField()
    duplicateScript = models.IntegerField()
```

Figura 4.6: Declaración de los objetos de Dr. Scratch: models.py

Una vez que ya sabe Dr. Scratch qué código debe ejecutar según lo visto ahora, procede y según la función, deberá realizar diversas acciones: consultas en la base de datos, realizar el análisis con Hairball, hacer una petición a otro servidor para obtener información, guardar el archivo... Todo este funcionamiento se irá detallando en la siguiente sección donde se explicarán las implementaciones realizadas.

Tras ejecutar dicho código y procesar toda la información, se debe mostrar una página de respuesta para lo que se utiliza el sistema de plantillas de Django llamadas templates. Estas plantillas son renderizadas con la información que se ha procesado y se quiere mostrar en dichas páginas.

Fichero log

Con el objetivo de tener un fichero que guarde información de todos los archivos analizados y poder observar en qué momento se producen problemas en nuestro servidor, se añadió un fichero log.txt a nuestro modelo. Este fichero es abierto, escrito y cerrado cada vez que un proyecto de Scratch es análizado agregando información relevante del mismo.

Base de datos

Dr. Scratch tiene para poder almacenar información una base de datos en MySQL 5.5.43 a la cual se accede para realizar consultas en las propias funciones con la ayuda de Django de un modo fácil y sencillo.

Para entender correctamente la estructura de la base de datos conviene fijarse en la *Figura* **??**.

Node.js (getSb2)

También se ha implementado con la ayuda del proyecto de un usuario de Github llamado Nathan[1] un servidor HTTP Node.js que se encarga de realizar peticiones al servidor de Scratch que será explicado más adelante.

Hairball y Kurt

El análisis del proyecto de Scratch se realiza mediante Hairball, el cual tenemos instalado en la máquina virtual y que se ayuda de Kurt para manipular proyectos complejos de Scratch con comandos sencillos de Python.

[1] `https://github.com/nathan/getsb2`

4.1.2. Capa de presentación: front-end

La capa de presentación de Dr. Scratch está formada por un conjunto de plantillas de código HTML que utilizan hojas de estilo CSS con la ayuda del framework Bootstrap. Estas páginas utilizan además, código JavaScript para mejorar su interactividad con el usuario y AJAX para realizar algunas funcionalidades en segundo plano.

En la *Figura* **??** podemos observar cuáles son las páginas de Dr. Scratch que son renderizadas y mostradas al usuario asociándolas a un recurso determinado.

El desarrollo de la capa de presentación ha sufrido numerosas modificaciones provocadas por las necesidades observadas en los usuarios en los test de usabilidad y por las peticiones realizadas directamente por varios docentes de todo el mundo. Dicha evolución de la plataforma se verá plasmada claramente en la siguiente sección.

4.2. Funcionalidades y evolución de Dr. Scratch

Con el objetivo de describir claramente cuál ha sido mi aportación al proyecto de Dr. Scratch quiero recordar que este proyecto es la continuación del Proyecto Fin de Carrera de Cristian Chusing que realizó la primera versión de Dr. Scratch, en la cual se había implementado la funcionalidad básica.

4.2.1. Versión Alpha

El prototipo inicial de Dr. Scratch debía implementar:

- Una página principal en la que poder indicar qué fichero quería subirse.
- Análisis del proyecto de Scratch mediante Hairball.
- Obtener toda la información útil del análisis de Hairball que quería mostrarse al usuario.
- Mostrar dicha información en un dashboard fácil de entender por el usuario.

Al comenzar este Proyecto Fin de Carrera mis tutores compartiron conmigo el código que Cristian les había hecho llegar, código en el cual la página principal tenía la apariencia mostrada en la *Figura 4.7* y que debía realizar dichas funciones perfectamente, porque tenía todo

programado, pero a la hora de ejecutar mostraba error y no conseguía mostrar la información del análisis de Hairball. Conociendo ahora completamente el funcionamiento de Dr. Scratch puedo suponer que el problema residía en que se debió realizar algún cambio en el plugin Hairball y el código que procesaba la información de Hairball se había quedado obsoleto.

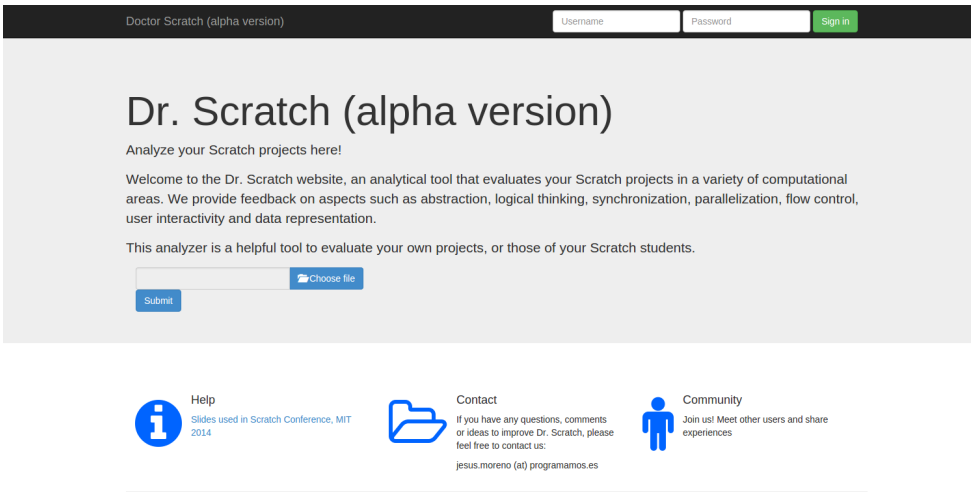

Figura 4.7: Apariencia de la versión Alpha de Dr. Scratch

4.2.2. Fase I

En este contexto, se fijaron mis principales objetivos a conseguir y mis primeras implementaciones que son las mostradas en la *Figura 4.8* .

Figura 4.8: Fase I del proyecto Dr. Scratch

Depuración del código

Puesta en producción

- Máquinas en la universidad
- Máquina virtual en Azure

4.2.3. Fase II

Figura 4.9: Fase II del proyecto Dr. Scratch

-

4.2.4. Fase III

Figura 4.10: Fase III del proyecto Dr. Scratch

-

Capítulo 5

Resultados

Capítulo 6

Conclusiones

6.1. Consecución de objetivos

Esta sección es la sección espejo de las dos primeras del capítulo de objetivos, donde se planteaba el objetivo general y se elaboraban los específicos.

Es aquí donde hay que debatir qué se ha conseguido y qué no. Cuando algo no se ha conseguido, se ha de justificar, en términos de qué problemas se han encontrado y qué medidas se han tomado para mitigar esos problemas.

6.2. Aplicación de lo aprendido

Aquí viene lo que has aprendido durante el Grado/Máster y que has aplicado en el TFG/TFM. Una buena idea es poner las asignaturas más relacionadas y comentar en un párrafo los conocimientos y habilidades puestos en práctica.

1. a
2. b

6.3. Lecciones aprendidas

Aquí viene lo que has aprendido en el Trabajo Fin de Grado/Máster.

1. a

2. b

6.4. Trabajos futuros

Ningún software se termina, así que aquí vienen ideas y funcionalidades que estaría bien tener implementadas en el futuro.

Es un apartado que sirve para dar ideas de cara a futuros TFGs/TFMs.

6.5. Valoración personal

Finalmente (y de manera opcional), hay gente que se anima a dar su punto de vista sobre el proyecto, lo que ha aprendido, lo que le gustaría haber aprendido, las tecnologías utilizadas y demás.

Apéndice A

Manual de usuario

Bibliografía

[1] R. G. Duque. Python para todos. pages 7–9, 2011.

[2] D. Greenfeld and A. Roy. Two scoops of django. 2013.

[3] M.-L. Liu and J. M. P. Sánchez. *Computación distribuida: fundamentos y aplicaciones*. Pearson Educación, 2004.

[4] M. Pilgrim and S. Willison. *Dive Into Python 3*, volume 2. Springer, 2009.

[5] S. Suehring. *Mysql bible*. John Wiley & Sons, Inc., 2002.