TRABAJO FIN DE GRADO

Desarrollo de un portal para una forja de proyectos de código abierto basado en el generador de sitios webs estáticos de Jekyll

ALBERTO ESTEBAN DÍAZ

2016

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Título: Desarrollo de un portal para una forja de proyectos de código abierto basado en el generador de sitios webs estáticos de Jekyll

Autor: Alberto Esteban Díaz

Tutor: Carlos Ángel Iglesias Fernández

Departamento: Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL

Presidente: D. Mercedes Garijo Ayestarán

Vocal: D. Tomás Robles Valladares

Secretario: D. Carlos Ángel Iglesias Fernández

Suplente: D. Amalio Francisco Nieto Serrano

Fecha de lectura:

Calificación:

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO DE INGENIERÍA DE SERVICIOS Y TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Desarrollo de un portal para una forja de proyectos de código abierto basado en el generador de sitios webs estáticos de Jekyll

ALBERTO ESTEBAN DÍAZ

2016

RESUMEN

Esta memoria plasma el resultado del desarrollo de un sistema para crear un portal a partir de los proyectos de una organización alojados en Github. El sistema se ha evaluado mediante el desarrollo de un portal web para la cuenta GitHub del Grupo de Sistemas Inteligentes (GSI) de la Universidad Politécnica de Madrid. Se pretende que esta memoria analice la tecnología de generación de sitios web estáticos Jekyll y evalúe sus posibilidades.

La arquitectura del proyecto consta de un módulo para que los administradores del sitio puedan seleccionar los proyectos que se publican, por lo que se han integrado facilidades de autenticación y autorización. La configuración del sistema se almacena en una base de datos en la nube, Firebase. Con el fin de facilitar la compatibilidad y adaptabilidad del sitio, se ha usado Jekyll-Bootstrap, y se ha personalizado al estilo del sitio web.

El resultado del proyecto está registrado como software de la UPM con licencia de código abierto.

Palabras clave: Jekyll, GitHub, Firebase, JavaScript, Bootstrap, jQuery JSON, YAML, Liquid, página web estática, base de datos en línea.

SUMMARY

This final work represents the results of the development of a system for creating automatically a web portal from the software projects hosted in a corporate GitHub account. This system has been evaluated through its application to the development of a website for the GitHub account of the Group of Intelligent Systems of Universidad Politécnica de Madrid. Herein it is intended to analyze the static web generator Jekyll and checking its features.

The proposed architecture of the project consists of a module that allows users to administer the web site, so that they can select the projects to be published. Thus, the project has integrated authorisation and authentication facilities. The system configuration is stored in a cloud database, Firebase. With the aim of enabling its cross-browser compatibility and adaptability, the framework Jekyll-Bootstrap has been chosen and has been customised to the web site theme.

The result of the project has been protected as UPM software and is available with an open source licence.

Keywords: Jekyll, GitHub, Firebase, JavaScript, Bootstrap, jQuery, JSON, YAML, Liquid, static web page, online database.

ÍNDICE DEL CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

1.1. MOTIVACIÓN

Internet es cada vez más grande y son más las posibilidades que nos ofrece. En los últimos años han aparecido multitud de aplicaciones y herramientas que facilitan el trabajo de los desarrolladores. Entre una de ellas se encuentra GitHub, una forja de proyectos de código abierto conocida mundialmente. Cada vez son más los usuarios u organizaciones que deciden almacenar sus proyectos en esta forja para aprovechar las ventajas ofrecidas, como el trabajo por ramas o el control de versiones. A su vez, consiguen que su trabajo sea reconocido y aporte valor.

Las cuentas de GitHub disponen de una página donde se detallan todos sus proyectos públicos. Esta página es idéntica para todas las cuentas, no permite personalización, y las posibilidades que ofrece no son muchas. Por tanto, todos los usuarios comparten el mismo estilo de página por defecto. No es posible filtrar los repositorios por fechas o cambiar la ordenación de los mismos.

En este proyecto se pretende desarrollar un sistema genérico de generación de sitios web corporativos que facilite su personalización y la capacidad de administrar su contenido.

El sistema se evaluará en la cuenta de GitHub del Grupo de Sistemas Inteligentes (GSI) de la UPM. Se busca dar la posibilidad de administrar los repositorios, permitir decidir cuáles mostrar, personalizar algunos aspectos y categorizarlos. Se conseguiría además diseñar un portal web distinto e innovador respecto a las páginas por defecto antes mencionadas.

Para la realización de la web se va a utilizar Jekyll, un generador de sitios webs estáticos. Los motivos de esta elección son varios. Se trata de una herramienta relativamente nueva, con ocho años de antigüedad, y no se ha trabajado antes con ella. Resultó interesante utilizar Jekyll para explorar su potencial y así conocer una tecnología adicional en la generación de páginas webs. Además, por el planteamiento inicial del proyecto, con una página estática sería suficiente para lograr los objetivos originales. Las carencias en cuanto al almacenamiento de datos se decidieron cubrir usando Firebase.

Firebase es otra de las herramientas que surgió hace pocos años y que está ganando terreno en el ámbito del almacenamiento online. Los motivos de esta elección son similares a los de Jekyll: no se ha trabajado antes con esta aplicación y su funcionalidad encaja con este proyecto. Este tipo de aplicaciones que permiten almacenar contenido en la 'nube' son cada vez más comunes y más usadas por los desarrolladores.

Uniendo estas tecnologías se pretende realizar un análisis de las ventajas e inconvenientes que supone su uso, al mismo tiempo que se diseña un portal web que resulte de utilidad al GSI.

1.2. OBJETIVOS

El objetivo principal es la elaboración de una herramienta web sencilla e intuitiva para mostrar los repositorios que el GSI tiene alojados en la forja GitHub. A lo largo del proyecto se han ido detectando una serie de objetivos secundarios para lograr esta finalidad:

- Entender el mecanismo de Jekyll y explorar sus posibilidades y limitaciones.
- Analizar el funcionamiento de Firebase y aprender a utilizarlo en un entorno web.
- Alojar la web en las Páginas de GitHub y comprobar su mantenimiento.
- Evaluar la información que es posible obtener de GitHub, buscar la forma adecuada de recuperarla y conseguir guardarla.
- Lograr una web adaptada a dispositivos móviles usando Bootstrap y hacerla compatible con todos los navegadores en su última versión.
- Profundizar en Javascript y explorar nuevas funcionalidades.

Como objetivo de fondo se busca diseñar un sistema dinámico y preparado para posibles modificaciones futuras. Cambiar la cuenta de GitHub o añadir nuevas categorías son algunos de los cambios a lo que la web debe estar preparada para hacer frente.

Se pretende además que este portal web mantenga el estilo de la web oficial del GSI, con el propósito de enlazar ambas páginas y mantener cierta uniformidad. En el desarrollo de esta memoria se irán planteando las modificaciones realizadas respecto del planteamiento inicial, con el cambio de objetivos que esto supone.

1.3. ESTRUCTURA DE LA MEMORIA

Una vez introducido el proyecto, sus objetivos y motivación, queda analizar el proceso de su desarrollo. En un primer lugar se detallarán las tecnologías, herramientas o sistemas utilizados. Se hará mayor hincapié en aquellos que no se han visto antes o que son menos comunes. Se buscará dar una visión global del contexto tecnológico de este proyecto señalando a su vez la aplicación particular de cada herramienta. Todo este contenido se verá bajo el título ‘Tecnologías habilitadoras’, en el Capítulo 2.

En el Capítulo 3, ‘Arquitectura’, se verá el funcionamiento global del portal web desarrollado. Al tratarse de una página estática, no existe un *back-end* como tal. Por este motivo se ha dividido la explicación de la parte de cliente en dos módulos: uno para la parte de administración donde se incluyen todas las conexiones con sistemas externos y configuración de la web y otro donde se describen las funciones de visualización que muestran los contenidos en la página. Este capítulo es fundamental para entender la lógica que sigue la web.

El resultado final se muestra en el Capítulo 4, ‘Caso de estudio’. El estilo y el aspecto de la web se muestran en este apartado. Por último se realizarán unas conclusiones sobre el proyecto realizado y se explicarán algunas mejoras y nuevas funcionalidades que se consideran adecuadas pero que aún no se han implementado.

2. TECNOLOGÍAS HABILITADORAS

2.1. INTRODUCCIÓN

En este capítulo se analizarán todas las herramientas y tecnologías que se han utilizado para el diseño del portal web. En primer lugar se explicarán los tres pilares del proyecto, Jekyll, GitHub y Firebase. Posteriormente se expondrán los lenguajes en que se ha desarrollado la aplicación web. Todas las secciones se centran en las funcionalidades utilizadas dentro del abanico de posibilidades que ofrecen.

2.2. JEKYLL

Jekyll [1] es un generador de portales web estáticos, escrito en Ruby y distribuido bajo una licencia de código abierto. Es además el motor que se encuentra detrás de las Páginas de GitHub (Sec. 2.3.2). Su principal característica es la ausencia del lado del servidor (*back-end*), determinante en el funcionamiento de toda página que se desarrolle usando esta herramienta.

Se presenta como una gema de Ruby y está alojado en un repositorio de GitHub. Tal y como se expone en la descripción de este repositorio, la filosofía de Jekyll es facilitar la creación de sitios webs personales donde lo que prime sea el contenido, huyendo de las configuraciones y complejidades innecesarias [2]. Como consecuencia de este planteamiento, no todas las aplicaciones se ajustan a las restricciones impuestas por Jekyll, principalmente en cuanto al manejo de datos y almacenamiento de los mismos.

La estructura de un proyecto realizado con Jekyll es similar a la de cualquier página web con algunas salvedades que se tratarán en la sección correspondiente (Sec. 2.2.3). Actualmente no existe ningún programa específico que facilite la gestión de los portales web realizados con Jekyll ni su alojamiento. Desde la web oficial de Jekyll se aconseja la utilización de una máquina con Linux, Unix o Mac como sistema operativo para trabajar con su herramienta, aunque también es posible utilizar Windows. Una de las opciones más extendidas en la web es la utilización de GitHub para alojar el portal web y la consola de Linux para realizar las actualizaciones de la página. Este es el método que se ha seguido para la elaboración del proyecto y se explicará más adelante.

Una vez que Jekyll ha generado el sitio web estático, el resultado son páginas con formato HTML, si bien es posible y necesaria la adición de funciones JavaScript y hojas de estilos CSS. Antes de la transformación que realiza Jekyll, los ficheros originales pueden estar en formato HTML o Markdown y hacen uso del lenguaje de marcado Liquid y YAML (Sec. 2.5).

2.2.1. HISTORIA DE JEKYLL

Tom Preston-Werner, fundador y ex CEO de la organización GitHub, fue el creador de Jekyll. En el año 2008, desde San Francisco, Tom escribió en su propio blog [3] (realizado con Jekyll) un artículo en el que anunciaba el nacimiento de la nueva herramienta y las nociones básicas para su comprensión. En este proyecto estuvieron implicados además varios diseñadores y programadores pertenecientes a su empresa. Crearon un repositorio público y una organización en GitHub de Jekyll para poder documentar y actualizar los cambios. Posteriormente desarrollaron la web oficial de Jekyll.

Con la partida de Tom Preston-Werner de GitHub en el 2014, el proyecto quedó en manos del resto de colaboradores. A pesar de tener una licencia de código abierto, GitHub siguió apoyando y mejorando el proyecto iniciado en 2008.

El nombre de Jekyll está relacionado con la novela “El extraño caso del Dr. Jekyll y Mr. Hyde” de Robert Louis Stevenson donde el doctor Jekyll crea una pócima que le transforma en Mr. Hyde, posible símil con la transformación que realiza este *software*.

Figura 2.1: Logo de Jekyll

2.2.2. JEKYLL BOOTSTRAP

Jekyll-Bootstrap (en adelante JB) [4] nos permite crear proyectos basados en Jekyll utilizando las funcionalidades de Bootstrap. Se trata de unas modificaciones realizadas a la instalación por defecto de Jekyll que añaden una serie de carpetas propias, integrando de esta manera la librería de Bootstrap en el proyecto. Es importante mantener la misma estructura para garantizar su buen funcionamiento.

El repositorio *jekyll-bootstrap* se encuentra en GitHub [4] y basta con clonarlo a nuestra cuenta para comenzar a desarrollar una web. Vienen predefinidas unas páginas de prueba donde se puede observar Bootstrap en funcionamiento. Usando la consola es posible desplegar esta web en local y comenzar a realizar modificaciones para adaptarla a los objetivos propios. Este es el proceso que se ha seguido en este proyecto.

La lógica interna es muy similar a la Jekyll. De hecho, aprovecha la estructura de Jekyll para añadir las mismas páginas por defecto incluyendo Bootstrap. De esta manera al utilizar JB se facilitan los primeros pasos si se quiere diseñar una web adaptada a dispositivos móviles. Además, al igual que Jekyll, es compatible con las Páginas de GitHub para su alojamiento. [5]

2.2.3. ESTRUCTURA DE CARPETAS

El repositorio base de JB tiene una estructura de carpetas predefinidas que se debe respetar. Inicialmente la web viene con el formato de un blog y por ese motivo algunas de las carpetas sólo tienen utilidad si el objetivo es diseñar un blog. A continuación se lista el contenido principal del repositorio inicial [6].

- **_config.yml**: archivo en formato YAML donde se almacenan datos de configuración de la página como los datos el autor, el resaltado de sintaxis o los archivos al generar la web. Para acceder a las variables aquí definidas se debe usar Liquid y el prefijo *site*.

- **_includes**: carpeta donde se almacenan fragmentos de código que pueden ser incluidos en las páginas HTML usando Liquid. JB utiliza esta característica para definir en esta carpeta los temas de la web que utilizan Bootstrap. Los temas por defecto de Jekyll incluyen los definidos en *_includes*. De esta manera al generar la página con Jekyll los temas predefinidos cargan automáticamente los definidos por JB y no es necesario realizar ninguna modificación del funcionamiento original. Todos ellos tienen una etiqueta, {{content}} en la cual se inyecta el código de las páginas que los utilizan. En la figura 2.2 se representa de manera esquemática el funcionamiento.

- **_layouts**: en esta carpeta están definidos los temas de Jekyll. Estos incluyen mediante Liquid los temas ya comentados de JB. Un mismo tema se puede y debe utilizarse en varias páginas para mantener los elementos comunes dependientes de un solo archivo. Los elementos comunes suelen ser la cabecera y el pie de página.

- **_posts**: carpeta donde se almacenan los posts publicados en la web. Son archivos con formato Markdown.

- **_drafts**: carpeta donde se almacenan los posts no publicados en la web. Tiene una estructura idéntica a la carpeta anterior. Ninguna de las dos ha sido utilizada en la web diseñada.

- **_site**: en esta carpeta es donde Jekyll genera la web y monta los archivos HTML resultantes. Todas las carpetas precedidas por un guion no son incluidas.

- **assets**: contiene las hojas de estilos, las imágenes de la web y las librerías de Javascript, entre las que se encuentran jQuery y Bootstrap por defecto para JB. Las funciones propias que se desarrollen deben estar incluidas en esta carpeta para asegurar que se incluyen en el sitio web generado.

- **index.md**: se trata de la página principal del portal web. Cuando Jekyll genera el sitio, transforma todos los archivos con extensión Mardown en HTML, incluyendo los fragmentos de *_includes* y los elementos del tema escogido de *_layouts*. Este archivo también puede estar escrito en HTML.

2.2.4. FRONT MATTER

Dentro de cada página es necesario incluir en la parte superior un fragmento de código YAML delimitado por tres guiones. En estas líneas se definen variables que pueden ser llamadas desde cualquier parte de la página usando Liquid. Esta región es lo que se conoce como Front Matter. Este término es fundamental para entender el funcionamiento de Jekyll.

Del mismo modo que para acceder a las variables del archivo *config.yml* se usa el prefijo *site*, para acceder a la variables definidas en el Front Matter se utiliza el prefijo *page*. Existen variables globales ya predefinidas que basta con asignar un valor para que Jekyll entienda cuál es la función de cada una. También es posible crear variables personalizadas y darles otros usos distintos.

En este proyecto se han utilizado las siguientes variables:

- **layout:** asigna un tema de la carpeta *_layouts* a la página.
- **title**: define el título de la página en el navegador y además se puede acceder a su valor usando el prefijo *page*, como ya se ha mencionado.
- **group:** se utiliza para crear grupos de páginas que pertenezcan a una misma categoría. El uso más típico es el de asignar un grupo común a todas las páginas que deban aparecer en el menú que posteriormente se genera con Liquid. En la sección 3.5.1 se detallará la creación del menú de la web.
- **order:** variable utilizada para ordenar las páginas del menú.
- **description:** esta variable es personalizada y se ha creado para poder acoplar elementos del menú bajo un desplegable.

Un ejemplo de Front Matter sería el siguiente:

```
---
layout: page
title: Agentes y Simulación Social
description: categories
group: navigation
order: 4
---
```

El código HTML de la página se situaría justo a continuación. La siguiente imagen muestra un ejemplo de cómo funciona JB al generar la página principal *index.md* con el tema *defaultJB* que contiene la cabecera y el pie de la página.

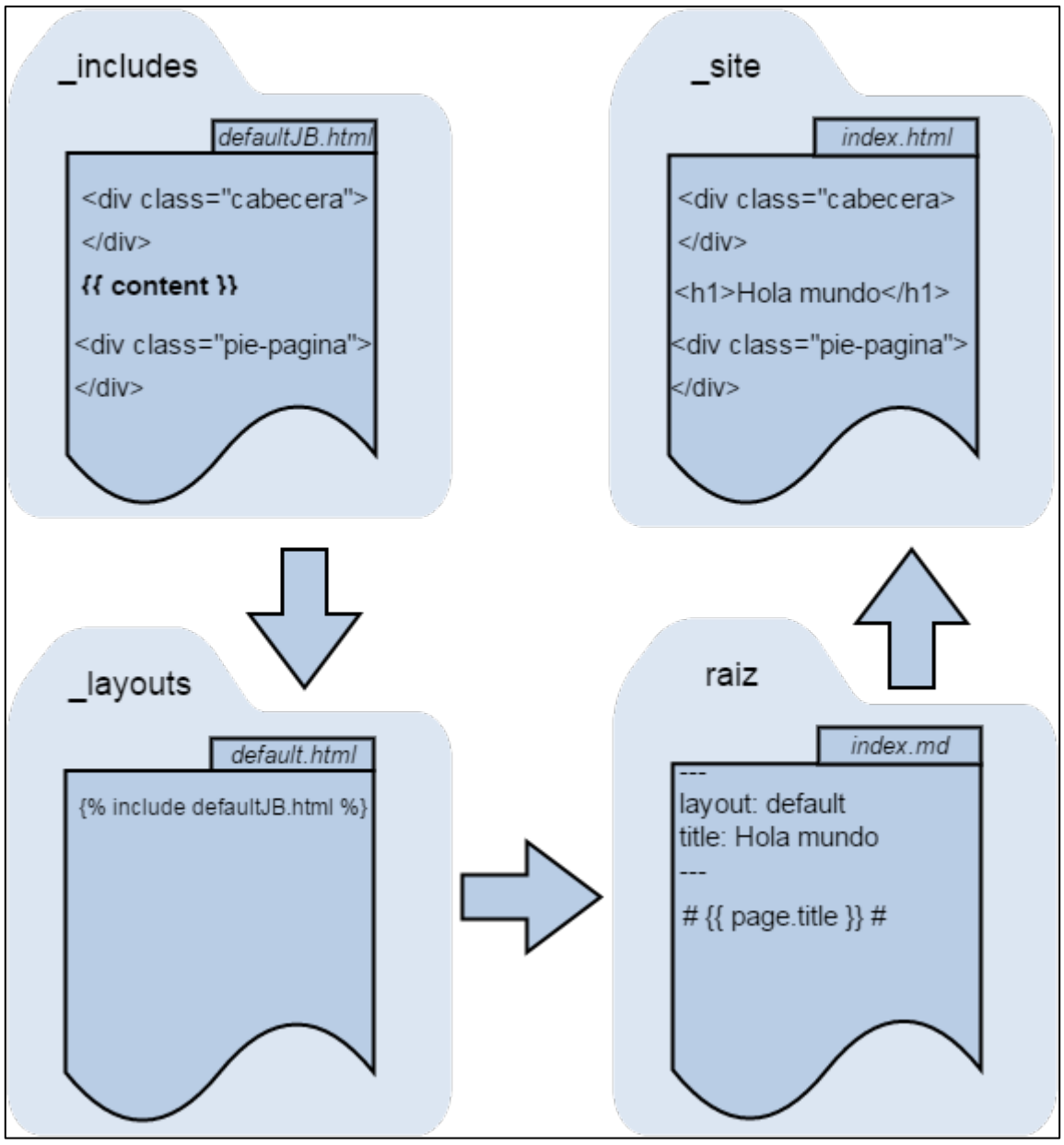

Figura 2.2: Esquema del funcionamiento de Jekyll-Bootstrap

De esta manera se llega a la conclusión de que el Front Matter es un elemento fundamental para la creación de páginas. Basta con definir el cuerpo de la web en un archivo *layout* y utilizarlo para el resto de las páginas.

2.2.5. MANEJO DE JEKYLL

Una de las principales ventajas que tiene Jekyll frente a otros mecanismos de generación de portales webs es su sencillez para el alojamiento y desarrollo si se utiliza la consola y las Páginas de GitHub. El primer paso es clonar el repositorio de JB a la cuenta de GitHub propia, a un repositorio con unas condiciones especiales (Sec.2.3.2). A partir de ese punto, para actualizar la web sólo es necesario subir el repositorio entero mediante el comando *commit*.

Además es posible ejecutar la web en local mediante el comando *jekyll serve* y accediendo a la misma por el navegador, en el puerto 4000 por defecto. Cada modificación del código que se realice se verá reflejada en el navegador al refrescar la página. También es posible cambiar esta configuración que actualiza los cambios en local sin necesidad de desplegar de nuevo la web.

Estas características hacen que el manejo de páginas realizadas con Jekyll y alojadas en GitHub sea sencillo y cómodo. El objetivo de un blog elaborado con esta herramienta es escribir los *posts* en la carpeta pertinente y subir el repositorio. No obstante, para este proyecto se quería huir de tener que estar actualizando la web usando la consola y ese es el motivo por el cual se buscó una base de datos externa, Firebase, que se verá en la sección 2.4

2.3. GITHUB

GitHub es una de las forjas de alojamiento de proyectos más utilizada a nivel mundial. Su política de código abierto le ha dado muchos seguidores y hoy en día se utiliza incluso como mecanismo de publicidad a nivel individual. La posibilidad de crear repositorios clones y modificar código de otros usuarios, el visor de ramas o el mantenimiento de varias versiones son algunas de sus funcionalidades más conocidas.

Para este proyecto se han utilizado dos características no tan comunes de GitHub: las Páginas de GitHub y su API. En los siguientes apartados se tratarán estos dos aspectos y en primer lugar se mostrará una manera de trabajar con GitHub desde equipos con el sistema operativo Windows.

2.3.1. GITHUB PARA ESCRITORIO

El manejo de repositorios almacenados en GitHub se suele llevar a cabo usando las consolas de Linux o Mac que vienen implícitas con sendos sistemas operativos. Si se busca utilizar el sistema operativo Windows existen varias alternativas. Una de las más simples es el uso de un software gratuito que instala en el equipo una aplicación visual para manejar los repositorios. La instalación incluye además una aplicación similar a las consolas de los dos sistemas operativos antes mentados. Se trata de *GitHub Desktop* (GitHub para escritorio) [7].

El terminal de Windows no es capaz de manejar el control de versiones Git que utiliza GitHub. Sin embargo, existe un paquete, *Posh-Git*, que instalado en la máquina permite usar los comandos de GitHub en la aplicación de Powershell de Windows. Este paquete se instala de forma automática con *GitHub Desktop* y proporciona una versión de Powershell ya definida para trabajar con GitHub. Incluye además un control del estado en que se encuentra el repositorio. En la siguiente imagen se muestra la interfaz de la consola de Windows de *GitHub Desktop.*

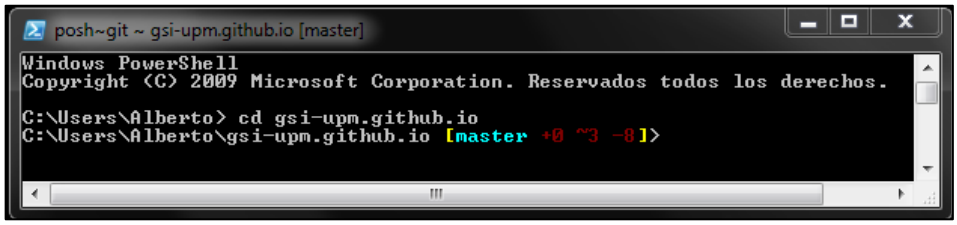

Figura 2.3: Git Shell - Consola de GitHub Desktop

Aunque la interfaz proporcionada permite manejar los repositorios de GitHub, para este proyecto se ha utilizado únicamente la consola de Windows *Git Shell*. El motivo de esta decisión ha sido por comodidad y por tener un control mayor sobre el estado del repositorio.

2.3.2. PÁGINAS DE GITHUB

Actualmente existen muchas maneras de realizar el alojamiento de páginas web de forma gratuita. Una de las más sencillas que se ha encontrado y totalmente compatible con Jekyll es utilizando la Páginas de GitHub (*GitHub Pages*) [8]. Se trata de una funcionalidad ofrecida por GitHub a través de la cual es posible alojar una web en un repositorio propio. Como requisito, el nombre del repositorio debe tener el formato *nombre-de-la-cuenta.github.io* y contener un archivo *index.html* como página principal, aunque es posible cambiarlo.

Tomando como ejemplo el proyecto desarrollado, la url de acceso a la web sería:

http://gsi-upm.github.io/

Si esta característica se junta con lo explicado en la sección 2.2.5 sobre el manejo de Jekyll, bastaría con clonar el repositorio de JB a un repositorio personal cuyo nombre cumpla el formato antes indicado. Esta es una de las principales ventajas de combinar Jekyll con este tipo de alojamiento. Como inconveniente, GitHub no permite la utilización de *plugins* de Jekyll. Por este motivo ha sido necesario desarrollar todas las funciones de la web partiendo desde el modelo básico de JB.

2.3.3. API DE GITHUB

GitHub pone a disposición de sus usuarios una serie de funciones predeterminadas que permiten acceder a la información de sus cuentas. Utilizando peticiones GET y pasando una serie de parámetros es posible recopilar datos de los repositorios de las respuestas en formato JSON. Además, si la cuenta es de una organización, también se pueden realizar peticiones de información acerca de los miembros y equipos que conforman la misma [9].

La librería jQuery de Javascript proporciona la función *getJSON*, capaz de realizar peticiones GET con AJAX y recoger la respuesta en un *callback*. En la siguiente tabla se muestran las peticiones típicas que se han utilizado en este proyecto. Para el nombre de la cuenta se ha tomado el de la organización del GSI. La diferencia entre una cuenta de usuario y otra de una organización radica en que la organización tiene equipos y miembros asociados.

Tabla 2.1: Peticiones de la API de GitHub

Petición	Respuesta
https://api.github.com/orgs/gsi-upm/repos?per_page=10	Los diez primeros repositorios
https://api.github.com/repos/gsi-upm/repoName?	Información del repositorio de nombre ‘repoName’
https://api.github.com/repos/gsi-upm/repoName/readme?	Datos del Readme del repositorio ‘repoName’
https://api.github.com/repos/gsi-upm/repoName/contributors?	Datos de colaboradores del repositorio ‘repoName’
https://api.github.com/orgs/gsi-upm/members?	Datos de miembros de la organización
https://api.github.com/orgs/gsi-upm/teams?	Datos de equipos de la organización
https://api.github.com/teams/idTeam/members?	Datos de los miembros del equipo con id ‘idTeam’

Además de las peticiones básicas existen unos parámetros que se pueden utilizar para aplicar condiciones a las peticiones. A continuación se exponen dos posibles usos:

- Combinando *per_page* y *page* se pueden realizar peticiones en grupos de repositorios, por ejemplo para evitar procesar toda la respuesta en un mismo instante. Estos dos parámetros son especialmente útiles para la paginación en una web o para realizar un ‘scroll infinito’, como el que se ha realizado en este proyecto.

- Al realizar una petición de los miembros de una organización es posible especificar el rol buscado, usuario o administrador. Con el parámetro *role* se puede seleccionar *user* o *admin* respectivamente. Esta práctica es necesaria si se busca discernir entre unos miembros y otros según sus privilegios, ya que en la información general de la respuesta de miembros no figura un campo que determine el rol. Por tanto, se deberían realizar dos peticiones, buscando un rol y después el otro.

2.3.4. PERMISOS DE ACCESO DE LA API

Las peticiones vistas en el apartado anterior sirven para acceder únicamente a la información pública de una cuenta de GitHub. Además, para peticiones sin identificar, GitHub limita el número de estas a 60 a la hora, y en caso de superarse devuelve un mensaje de error. Cada petición va asociada a la IP del equipo desde donde se realiza.

Para poder acceder a toda la información completa de una cuenta y sin limitaciones, GitHub plantea una posibilidad: dar permisos a una aplicación desde la cuenta de GitHub. Usando la configuración de la cuenta se puede registrar una aplicación para dar permisos de lanzamiento de peticiones. Una vez registrada usando la url de la web, GitHub proporciona un identificador único y una clave. Usando ambos y siguiendo unos pasos, se puede solicitar una clave especial que sirva de parámetro en las peticiones ya vistas. Esta clave es lo que se conoce como *token* y es lo que se debe pasar en las peticiones como valor del parámetro *access_token*.

De esta manera es posible lanzar hasta 5000 peticiones a la hora y tener acceso a toda la información pública y privada de una cuenta. A todas las peticiones anteriores se les debe añadir este nuevo parámetro. GitHub detecta cuándo un *token* es almacenado dentro de un repositorio y lo bloquea tras notificar al propietario. Por este motivo el *token* debe estar almacenado en una base de datos externa a GitHub.

2.4. FIREBASE

Firebase [10] es un servicio online gratuito que proporciona un *back-end* para aplicaciones web y móviles. Se trata de una base de datos remota, en formato JSON, capaz de ser una fuente de datos en tiempo real. Además proporciona un servicio de autenticación de usuarios y librerías para trabajar con sus funciones.

Fue fundada en septiembre de 2011 y a finales del año 2014 Google adquirió a esta *startup*. Según informó el CEO de Firebase en el blog oficial, se unieron a Google con el objetivo de llegar a más usuarios, aumentar sus posibilidades y fusionar tecnologías complementarias, como el servicio de almacenamiento en la nube de Google. Actualmente más de 400000 programadores utilizan este servicio.

Las conexiones en Firebase se realizan usando *sockets*, lo que aporta rapidez a la transmisión de datos. Una página que utilice Firebase mantiene una conexión bidireccional constante con capacidad de escuchar los cambios que se produzcan. Este aspecto convierte a Firebase en una herramienta útil en la elaboración de blogs o incluso en un servicio de mensajería instantánea (*chat*).

Figura 2.4: Logo oficial de Firebase

En este proyecto se ha utilizado la librería Firebase.js de JavaScript. En la página oficial de Firebase existe documentación detallada sobre las funciones de esta librería necesarias para la comunicación entre la página web y la base de datos remota. Basta con incluir la librería en la referencias de la web para comenzar a utilizar esta herramienta.

La adhesión de Firebase a Google hizo posible usar la cuenta de Google para registrarse en Firebase y comenzar a desarrollar la aplicación. El control sobre la base de datos se realiza vía online a través de una página web accesible con el usuario propio. Actualmente existe un plan gratuito simple y planes más complejos de pago mensual que incluyen un mayor número de conexiones simultáneas, usuarios ilimitados, mayor capacidad de almacenaje y un dominio personalizable.

La utilización de Firebase es totalmente compatible con Jekyll, ya que Firebase sólo afecta a la funcionalidad de la web y no a cómo se genera. De hecho, en la web se aconseja complementar páginas de Jekyll son servicios de base de datos online, como este o como MongoDB. En el capítulo de Arquitectura, sección 3.3.1 se profundizará en las funciones de esta herramienta.

2.5. TECNOLOGÍAS DE PROGRAMACIÓN

En este apartado se van a exponer brevemente las tecnologías y lenguajes trabajados, al igual que su contexto dentro de este proyecto.

2.5.1. LIQUID

Liquid [11] es un lenguaje de plantillas de código abierto escrito en Ruby cuyo fin es cargar contenido dinámico. Su aplicación se extiende a muchos sistemas entre los que se encuentra Jekyll. En este contexto Liquid se utiliza para modificar el código HTML en función de variables o de la propia estructura de la web. El código de Liquid se puede categorizar en objetos, etiquetas y filtros.

- **Objetos**: muestran contenido en la página. Se marcan con dos llaves y pueden mostrar texto o el contenido de variables. En el caso de Jekyll, los objetos de Liquid se utilizan para acceder a las variables del archivo de configuración y a las del Front Matter. Al generar los archivos de HTML, Jekyll transforma los objetos en texto plano, por lo que es necesario complementarlo con etiquetas de HTML.

```
<p>{{ site.page }}</p> → GSI UPM GitHub
<p>{{ 'Texto literal' }}</p> → Texto literal
```

- **Etiquetas**: se utiliza para aplicar lógica a la plantilla. Se marca utilizando llaves y porcentajes. Pueden implementar bucles, condiciones, *switchs*, asignar variables o incluir fragmentos de código. El código HTML se genera en función de los resultados de estas operaciones lógicas.

```
<title>
{% if page.title %}
    {{ page.title }}
{% else %}
    {{ site.title }}
{% endif %}
</title>
```

- **Filtros**: modifican el contenido que se muestra utilizando objetos o etiquetas. Existen diversos filtros que pueden verse en la documentación de Liquid. En el siguiente ejemplo guarda en una variable el conjunto de página existentes en un proyecto de Jekyll ordenadas por la variable 'order' del Front Matter de cada una.

```
{% assign sorted_pages = site.pages | sort:"order" %}
```

2.5.2. YAML

YAML (*YAML Ain't Another Markup Language*, 'YAML no es otro lenguaje de marcado') es un estándar de serialización de datos legible con implementación en todos los lenguajes de programación. Es uno de los estándares de representación de datos más extendido en el mundo, junto con XML y JSON. Es adecuado para sistemas que impliquen labores de un humano, ya que su estructura es simple. Uno de los usos más comunes de YAML es para ficheros de configuración, como es el caso de Jekyll [12].

Si el fichero tiene formato *yml* no es necesario marcar en el contenido que se está utilizando YAML, pues el propio fichero lo marca. Es el caso del fichero *_config.yml* de Jekyll. Si por el contrario se busca introducir un fragmento de YAML en un archivo con otra extensión, se debe delimitar con tres guiones consecutivos. Este es el caso del Front Matter ya explicado.

En este lenguaje la indexación tiene significado de pertenencia a un nodo superior, por lo que es más estricto que otros lenguajes. Para acceder a estas variables se utiliza Liquid como se muestra en el siguiente ejemplo del fichero de configuración del proyecto de Jekyll.

```
author :
  name : Grupo de Sistemas Inteligentes
  github : gsi.gsi-upm
  university : Universidad Politécnica de Madrid
                    {{ site.author.university }}
```

2.5.3. JSON

JSON (JavaScript Object Notation) es un formato de intercambio de datos ligero [13]. Su sintaxis es sencilla y además es fácil de analizar y generar para las máquinas. Una de sus ventajas frente a XML o YAML es justo la facilidad de crear un analizador. Actualmente es muy utilizado en aplicaciones que involucren JavaScript y AJAX, por la existencia de funciones predeterminadas capaces de analizar la respuesta de una petición AJAX en formato JSON.

En este proyecto se ha utilizado JSON para extraer la información de las respuestas a las peticiones a GitHub. También se ha utilizado para almacenar los datos en Firebase, aunque para este último se ha usado otras funciones de guardado, haciendo innecesario transformar el formato del contenido a guardar a JSON.

Su estructura es similar a la de YAML añadiendo algunos detalles particulares como las llaves, las comas al final de cada elemento o las comillas en los campos. El siguiente ejemplo muestra un objeto con sus atributos en formato JSON

```
[{
  "id": 50830157,
  "name": "gsi-upm.github.io",
  "full_name": "gsi-upm/gsi-upm.github.io",
  "owner": {
     "login": "gsi-upm",
     "id": 2894736
  },
  "private": true
}]
```

2.5.4. MARKDOWN

Se trata de un lenguaje de marcado ligero utilizado para dar formato a textos web. Sirve como una alternativa simple a HTML, aunque sus posibilidades son mucho más limitadas. El objetivo de Markdown era facilitar a los usuarios un mecanismo de dar formato a sus páginas sin necesidad de entender de HTML.

Jekyll es capaz de entender páginas escritas tanto en HTML como en Markdown y generar la página estática final en HTML. Esta funcionalidad de Jekyll ofrece la posibilidad de crear páginas directamente en Markdown, siguiendo su filosofía de que el contenido es lo que prima. Para este proyecto no se ha utilizado esta característica. Aunque el formato de la página sea de Markdown (*.md*), el contenido puede ser tanto Markdown como HTML.

Existen programas que transforman en tiempo real del código Markdown y muestran el resultado, como MarkdownPad. En la siguiente imagen se muestra la interfaz del programa con los principales elementos de Markdown y su representación.

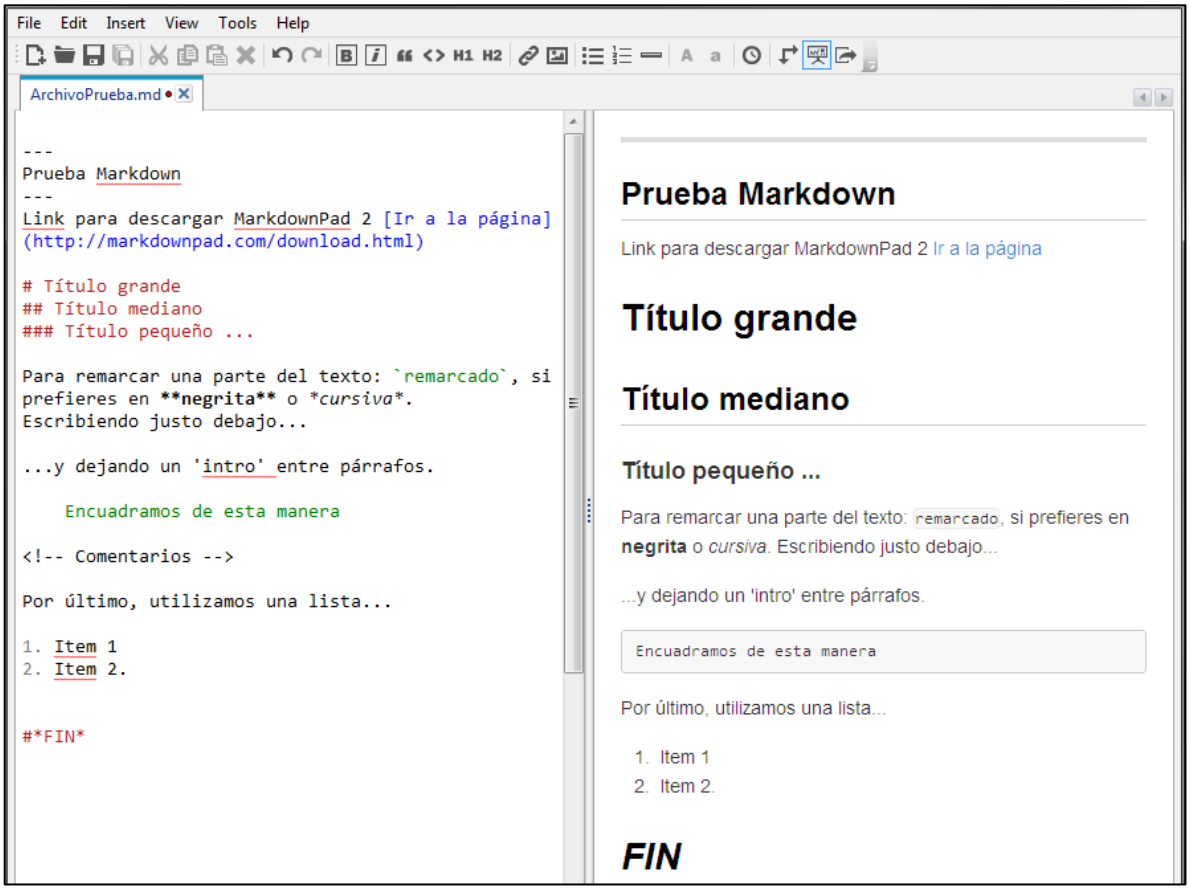

Figura 2.5:Interfaz MarkdownPad

Este formato es además el que se utiliza en muchos de los archivos *Readme* de los repositorios GitHub. En el capítulo de arquitectura se verá cómo se ha logrado llegar a mostrar el contenido de estos archivos en la web desarrollada.

2.5.5. HTML, CSS Y JAVASCRIPT

El portal web se ha desarrollado en HTML y aportando estilo con CSS3. JavaScript se ha utilizado para el funcionamiento de la web en el lado del cliente. La combinación de estos tres elementos es el pilar que soporta el portal.

De HTML se han utilizado elementos típicos como los contenedores, las listas, botones, cajas de texto o títulos. Prácticamente todos estos elementos tienen clases predefinidas en Bootstrap que provocan el efecto *responsive*. Es posible modificar estas clases siempre que no se alteren sus propiedades de posicionamiento, ya que se podría perder la efectividad de Bootstrap.

Existen una serie de librerías de JavaScript que se han incluido en este proyecto. Se ha buscado no provocar incompatibilidades entre ellas. A continuación se listan y se explican brevemente:

- **jQuery:** es una librería fundamental que facilita el trabajo con las funciones de JavaScript. Permite realizar peticiones AJAX y evaluar las respuestas desde cliente. Además es capaz de interactuar con los elementos HTML insertando o eliminando contenido. Estas dos funcionalidades han sido las más utilizadas en este proyecto.

- **Bootstrap:** ya se ha mencionado la utilidad de esta librería dentro de las páginas web. Es necesario incluirla para su correcto funcionamiento interno. Tanto para Bootstrap como

para jQuery existen versiones compactas de las librerías que reducen considerablemente el tamaño de las mismas. Se marcan con la etiqueta *min* al final del nombre de la librería.

- **Firebase:** también se ha explicado en su sección correspondiente la utilidad de esta otra librería. En este caso sí se utilizan las funciones internas para el manejo de la base de datos.

- **Rundown y Showdown:** se trata de dos librerías capaces de transformar el código de formato RDoc o Markdown respectivamente a HTML. De Rundown existe poca información y fue necesario explorar el código de la librería para entender su funcionamiento y ver las funciones que se deben utilizar [14]. Por el contrario de Showdown existe mucha más información en la web y con sólo la información del proyecto Showdown en GitHub es posible entender su mecanismo y empezar a utilizarlo [15]. En el capítulo de Arquitectura, sección 3.4.3 se analizará la utilidad de estas dos librerías en la descodificación de los archivos Readme.

- **Bootstrap Toggle y Bootstrap Datetimepicker:** estas dos librerías aportan dos elementos adicionales no incluidos en la librería de Bootstrap. El primero altera la apariencia del elemento *checkbox* [16] y el segundo proporciona un selector de fecha con formato de calendario [17]. Ambos vienen con una hoja de estilos predefinida. En la siguiente figura se puede observar el resultado de ambos elementos.

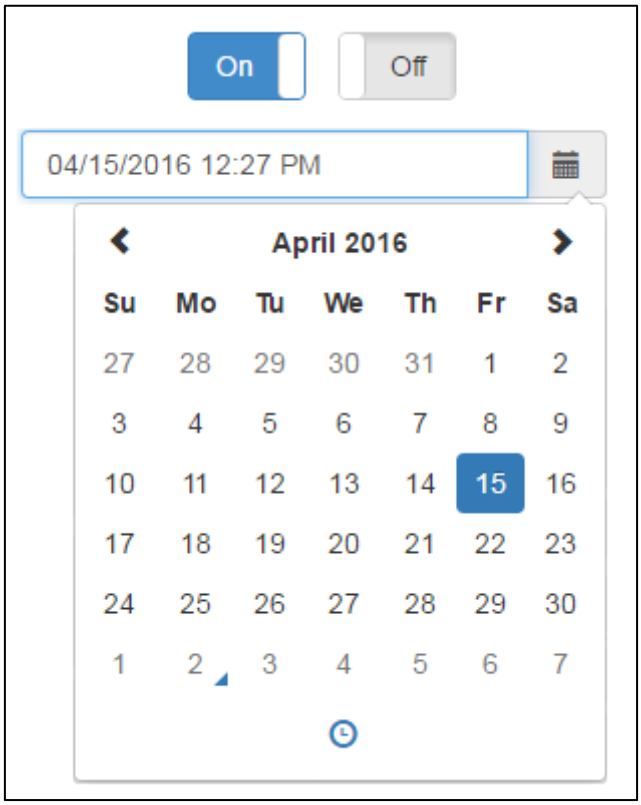

Figura 2.6: Elemento Toggle y DateTimePicker de Bootstrap

2.6. RESUMEN

En este capítulo se han definido las tecnologías y sistemas que han hecho posible el desarrollo de este proyecto. Se ha visto que Jekyll, GitHub y Firebase son los tres términos fundamentales sobre los que se construye el portal web y se ha llegado a la conclusión de que, a pesar de ser un sitio web generado con Jekyll, el resto de elementos son los típicos utilizados en el desarrollo web (HTML, CSS y JavaScript). En el siguiente capítulo se analizará la arquitectura del proyecto y su funcionamiento.

3. ARQUITECTURA

3.1. INTRODUCCIÓN

El portal web que se ha desarrollado está compuesto por los elementos típicos de una página: HTML, CSS y JavaScript. En esta sección se analizará la forma en que se relacionan y se verá el funcionamiento interno de la página. Además se verán con detalle todas las funciones JavaScript desarrolladas que soportan la estructura de la web. Se examinarán los mecanismos de conexión con GitHub y Firebase.

La propuesta inicial era almacenar el contenido en un fichero de formato JSON junto con los demás archivos del proyecto. De esta manera se podría acceder a la información tan sólo leyendo dicho fichero. Sin embargo, para esta opción sería necesario utilizar PHP o un lenguaje similar para interactuar con el servidor donde estuviera alojado y realizar peticiones. Este es el funcionamiento habitual de las páginas web con bases de datos. Se descartó por dos motivos:

1. Las páginas generadas con Jekyll son estáticas, lo que quiere decir que si utilizamos PHP o similares estaríamos perdiendo la esencia de esta herramienta y las ventajas que nos ofrece, como mayor velocidad al ejecutarse todo en el lado del cliente o la poca complejidad en el tratamiento de los datos.
2. El portal web está alojado en GitHub y éste no permite la utilización de PHP [18].

Se escogió Firebase por no haber trabajado nunca con esta herramienta, por ser gratuito y por ser adecuado para este tipo de aplicaciones: *back-end* de páginas estáticas que además proporciona autenticación [7]. De esta manera toda la información se recupera en el lado del cliente (*front-end*).

La elección de GitHub como alojamiento fue consecuencia del estudio previo sobre las ventajas de la utilización de esta herramienta. Al igual que Firebase, es gratuito y su funcionamiento es como el de cualquier repositorio de GitHub: todas las modificaciones se realizan mediante la consola. Como contrapartida, la imposibilidad de usar PHP ha provocado que se hayan tenido que buscar soluciones alternativas que se irán viendo en el desarrollo de esta sección.

En un principio se mostrará la arquitectura general de la web dividida en dos módulos que establecen conexiones con GitHub y Firebase. Posteriormente se verá cada uno de los módulos por separado con sus funciones, su lógica y sus posibilidades. Por último de forma general se examinarán las partes comunes a ambos módulos como el mecanismo del Front Matter de Jekyll y el diseño con Bootstrap.

3.2. ARQUITECTURA GENERAL

En la siguiente figura se muestra un esquema del funcionamiento global de la web. Los tres bloques fundamentales son la propia aplicación web GSI UPM GitHub, Firebase y GitHub. La comunicación entre ellos se detalla debajo y más adelante se profundizará en cada módulo por separado.

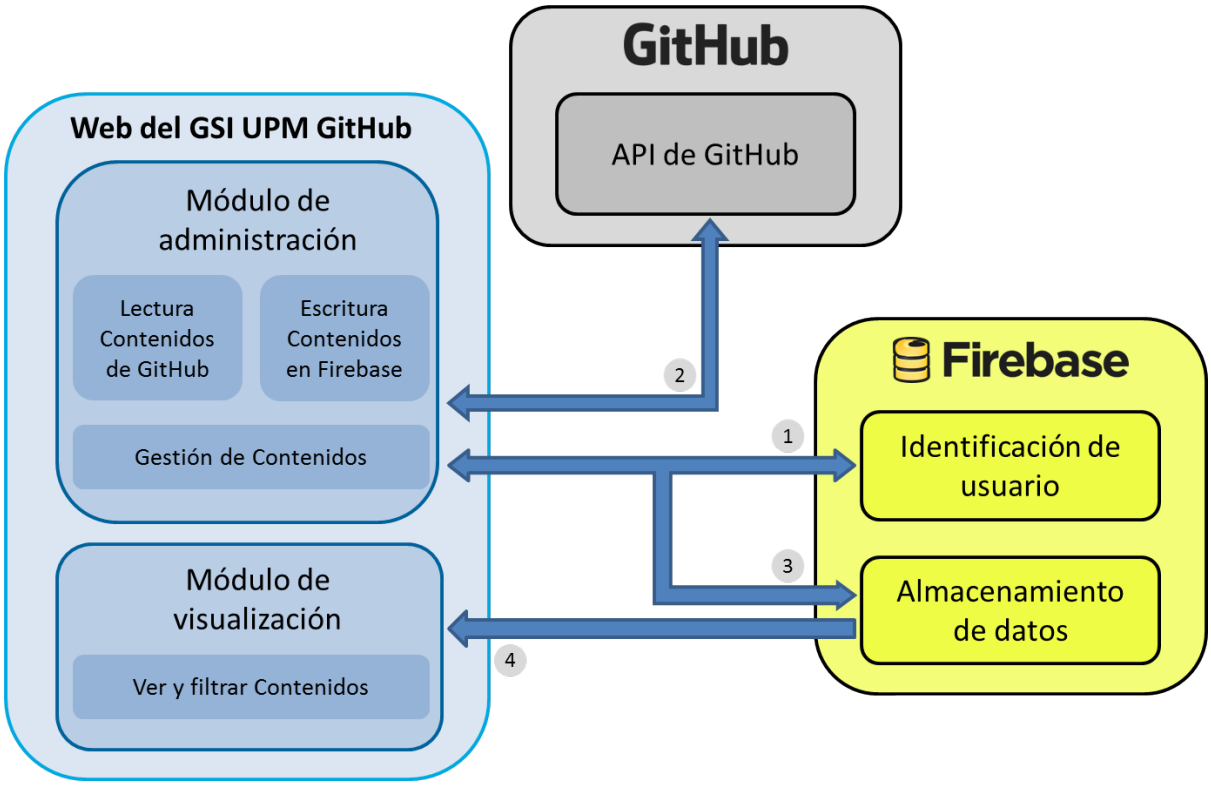

Figura 3.1: Esquema global

Ambos módulos dentro de la web se corresponden con el lado del cliente; GitHub y Firebase podrían verse como el lado del servidor, el primer caso para petición de información de la cuenta de GitHub 'gsi-upm' y el segundo para almacenar datos y la autenticación. La numeración indica el proceso lógico diseñado para el proyecto:

1. **Credenciales de administrador:** para poder acceder a la página de administración de la web es necesario identificarse con una cuenta de administrador. Se ha utilizado la autenticación y el control de sesiones que propone Firebase.

2. **Carga de los repositorios de GitHub:** desde la página de administración se realiza una petición AJAX a GitHub para obtener la información de los proyectos pertenecientes a la cuenta 'gsi-upm'. La respuesta viene dada en formato JSON y es procesada en el lado del cliente. Este proceso se corresponde con 'Lectura Contenidos de GitHub' dentro del módulo de administración de la web y se realiza de manera automática.

3. **Configuración de los repositorios:** el administrador de la web puedes seleccionar varias opciones relativas a los repositorios y dichos ajustes se guardan en Firebase junto con la información específica de cada proyecto. Todo ello se guarda también en formato JSON. El administrador es el que ejecuta estos procesos de manera manual. Se corresponden con los dos módulos, 'Escritura Contenidos en Firebase' y 'Gestión de Contenidos'.

4. **Visualización pública:** los usuarios que entren en la web tendrán acceso a los repositorios que el administrador haya marcado como públicos. Esta información se obtiene de Firebase, utilizando una librería de JavaScript que facilita la transmisión de información de una manera rápida. Tendrá además acceso a información sobre los miembros de la organización *gsi-upm* y los equipos guardados en el paso anterior.

Con este esquema de procesos se logra que el contenido de la web se obtenga directamente de Firebase y se mantenga inalterado hasta que el administrador actualice la información usando el módulo de administración. De esta manera, se suplen las carencias de Jekyll usando Firebase como base de datos propia.

3.3. MÓDULO DE ADMINISTRACIÓN

Todas las funciones de control y mantenimiento de la web pertenecen a este módulo. Para poder acceder al mismo es necesario introducir los credenciales de administrador en la web. Firebase tiene una funcionalidad que permite crear usuarios y contraseñas que posteriormente se pueden utilizar como método de autenticación en una web, junto con una serie de funciones de su librería.

3.3.1. CONEXIÓN CON FIREBASE

Firebase proporciona una librería de JavaScript, *Firebase.js*, con las funciones necesarias para establecer conexión y acceder al contenido. En su web oficial explican brevemente cómo utilizarlas [4]. Las funciones utilizadas para este proyecto utilizadas se pueden dividir en tres bloques principales:

- **Conexión:** se debe crear un objeto Firebase de JavaScript para seleccionar la url de la base de datos propia:

```
var nameBBDD = "https://shining-torch-549.firebaseio.com/";
var myDataRef = new Firebase(nameBBDD);
```

El objeto 'myDataRef' tiene una serie de funciones asociadas que permiten interactuar con el contenido almacenado. Sólo es necesario definirlo como una variable global y ya se puede utilizar en todo el código. Más adelante se verá que en muchas ocasiones es necesario crear objetos Firebase temporales para referirse a los nodos existentes en la base de datos.

- **Autenticación del usuario:** desde el panel de Firebase se definen las cuentas de usuarios y contraseñas de acceso. Las funciones que interactúan y controlan la sesión son:

```
//Inicio de sesion
myDataRef.authWithPassword({
                email    : “user”,
                password : “password”
        }, function(error, authData) {
                if (error == false){
                        //Codigo si el usuario es correcto
                }
        },{remember: "sessionOnly"}
);
```

El parámetro 'sessionOnly' activa el cierre de sesión cuando la página de la web se cierra. La duración de la sesión se define en el panel de Firebase.

```
//Comprobar si hay sesión iniciada
var authData = myDataRef.getAuth();

//Cerrar sesion
myDataRef.unauth();

//Controlar los cambios de estado de sesion
myDataRef.onAuth(function(authData) {
	//Codigo si ha cambiado el estado de la sesion
});
```

- **Manejo de los datos:** la información se guarda en Firebase en formato JSON. Las tres funciones principales que se utilizan son la de guardar datos, recuperarlos y actualizar los ya existentes. Cada entrada o nodo JSON se puede entender como una tabla y los hijos de ese nodo los registros. Para manejar los datos se debe instanciar el hijo (o tabla) al que se refiere. La función .val() en este contexto nos devuelve un array con los objetos de las respuesta [5].

```
//Recuperar datos del nodo "child"
refTemp = new Firebase(nameBBDD + "child");
refTemp.on("value", function(snapshot) {
	var childData = snapshot.val();
	$(childData).each(function () {
		//Codigo para cada entrada
	});
});

//Guardar datos en nodo "child" con id "repos_0"
var reposRef = myDataRef.child("repos");
reposRef.child("repos_0").set({
	id: "id_repos_0"
	name: "name_repos_0"
});

//Actualizar datos en nodo "child" del id "repos_0"
//Solo actualiza los datos dentro del 'update'
reposRef.child("repos_0").update({
	name: "name_repos_0"
});
```

3.3.2. PROCESO DE IDENTIFICACIÓN DEL ADMINISTRADOR

Existen numerosas formas de diseñar un control de usuario para páginas web. Para decidir cuál de ellos utilizar se plantearon varias opciones: un fichero .htaccess, control desde un servidor propio, servicios de pago comoAuth0, o la identificación de Firebase. En un principio se desarrolló el acceso mediante un fichero .htaccess pero debido a las carencias del mismo en cuanto a control de sesiones y usuarios se decidió buscar otro. Los métodos de pago también se descartaron y la idea de usar un servidor físico no encajaba con la filosofía de Jekyll.

La autenticación de usuarios de Firebase cumplía con todos los requisitos: gratuita, integrada con la base de datos online utilizada, sencilla de implementar y adaptable para un control de sesiones.

Además existe una documentación amplia y detallada sobre la utilización de las funciones de Firebase y en concreto de esta funcionalidad.

En el archivo *login-functions.js* se encuentran todas las funciones desarrolladas que controlan el acceso y la sesión del administrador. Se ha creado una única cuenta y se ha establecido la duración de la sesión a 20 minutos. Estos ajustes se pueden modificar en cualquier momento desde Firebase. Las funciones JavaScript implementadas son:

- ***loginUser(form)***: recibe el formulario de identificación (para extraer el nombre de usuario y contraseña) y verifica si son correctos usando Firebase. Si el usuario es correcto llama a la página *admin.html*; si no lo notifica. Para ello se utilizan las funciones de Firebase que se han detallado anteriormente para la autenticación de usuarios.

- ***sessionControl()***: cada vez que se carga una página esta función se ejecuta y si hay sesión iniciada activa los controles de administración en la barra del menú. Dichos controles se mantienen en todas las páginas de la web, ya que modifican el menú, común a todas a las páginas**.**

- ***logoutUser()***: cierra la sesión y si la página activa es la de administración, sale de la misma y llama a la principal (*index.html*). Elimina los controles de administración. Si se cierra sesión desde una página que no sea la de administración se mantiene en la misma página y lo notifica con un aviso.

- ***accessAdmin()***: se ejecuta cada vez que se llama a la página *admin.html*. Su objetivo es evitar acceder a la página de administración usando otros métodos que no sean el formulario de identificación, como puede ser escribir la url en el navegador. En caso de tratar de acceder por uno de estos métodos, redirige a la página principal de la web, siempre y cuando no esté identificado.

- ***adminSession()***: activa un proceso para escuchar los cambios en el estado de la sesión del usuario en Firebase. Se arranca cada vez que se entra en una página y el administrador tiene sesión iniciada. Controla si la sesión expira y en dicho caso lo notifica y llama a la página principal en caso de estar en la de administración.

Con estas cinco funciones JavaScript el acceso a la administración de la web queda protegido y la sesión controlada. Este mecanismo es diferente al de cualquier otro sistema al realizar todas las comprobaciones de sesión desde el cliente y en concreto desde JavaScript. Sin embargo, la finalidad de este proyecto no es la de un sistema de identificación de usuarios potente y con muchas posibilidades sino que es una necesidad que se ha cubierto de una forma sencilla.

En la siguiente figura se muestra un diagrama UML del funcionamiento de esta parte de la web.

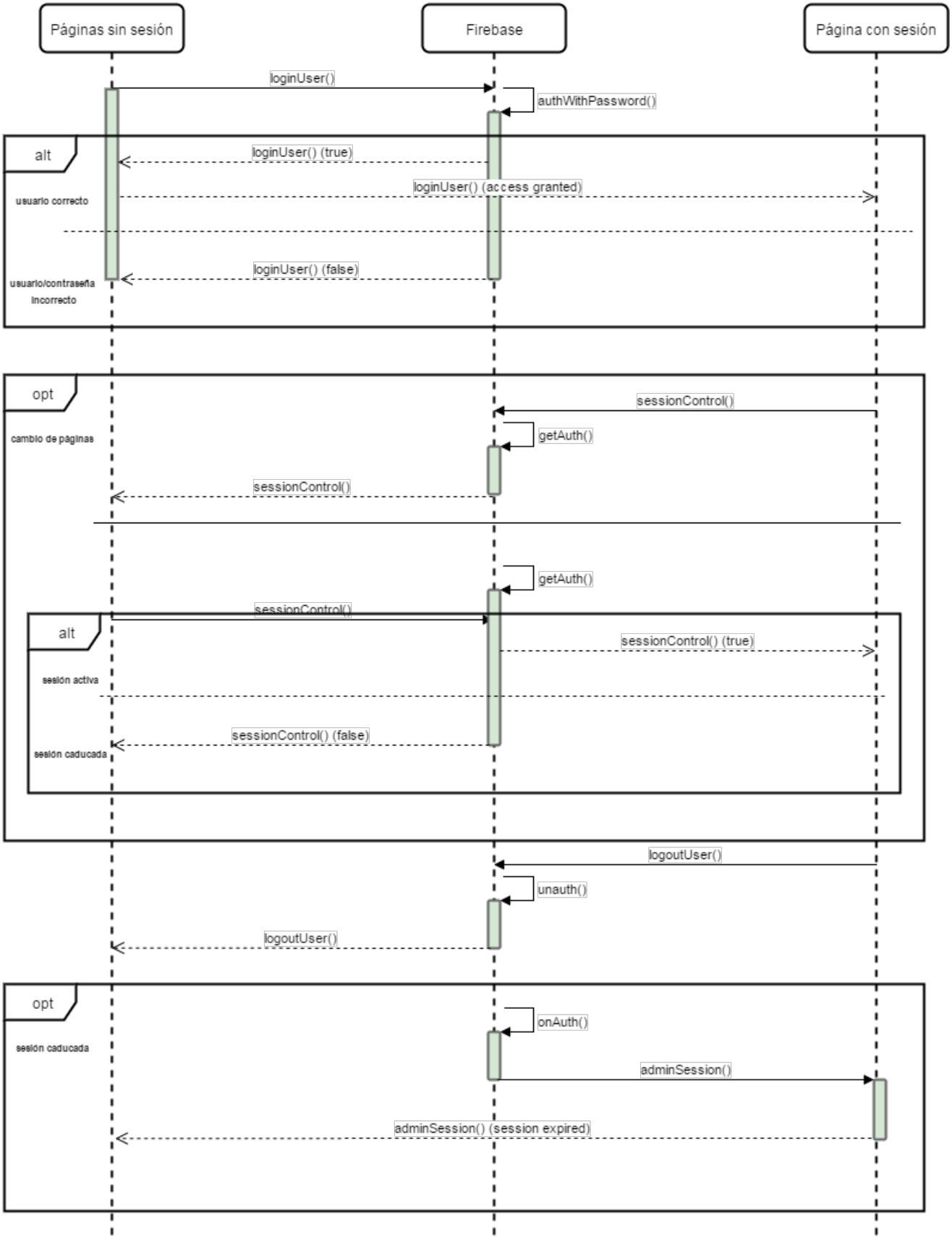

Figura 3.2: Esquema identificación del administrador

3.3.3. LECTURA DE CONTENIDOS DE GITHUB

Para conseguir el contenido buscado en GitHub, se encontraron dos formas de proceder:

- **Liquid:** GitHub permite el acceso a información de los repositorios de una cuenta si se utiliza Liquid en una página construida con Jekyll y alojada en GitHub [5]. Sin embargo, este mecanismo tiene una característica que no encajaba con el objetivo de la web: la información de los repositorios sólo se actualiza cada vez que se sube el proyecto de la web a GitHub. Esto significa que la web siempre mostraría el contenido actualizado al día en que se subió la web a GitHub. Además, se perdía la posibilidad de gestionar el contenido. A pesar de ser la opción idónea en un principio, esta peculiaridad provocó su descarte.
- **API de GitHub:** mediante peticiones GET se puede acceder a todo el contenido. Este procedimiento no tiene limitaciones, permite acceder a más información aparte de los repositorios y el contenido está actualizado. El inconveniente es la necesidad de utilizar JavaScript y AJAX para toda comunicación con GitHub. Este fue el método elegido para obtener los datos de los repositorios.

En el apartado de GitHub dentro del capítulo 'Tecnologías habilitadoras' se explicó la necesidad de registrar la web como una aplicación permitida en los ajustes de la cuenta del GSI de GitHub (Sec. 2.2.4). También se llegaba a la conclusión de que, una vez realizados una serie de pasos, se podía conseguir un *token* que daba permisos de lectura a todos los contenidos, públicos y privados.

Las peticiones AJAX, usando la librería jQuery y su función *getJSON*, son el mecanismo de acceso a la información necesaria de GitHub (Sec. 2.3.3). Se trata de peticiones *GET HTTP* que utilizan la url de la API de GitHub como parámetros de entrada y la respuesta se devuelve en una variable JSON que es necesario analizar para su correcta lectura. En todas las peticiones que se realizan es preciso pasar el *token* obtenido como parte de la url, aparte de los demás parámetros que ya se explicaron. A continuación se muestra un fragmento del resultado de una petición de información del propio repositorio de esta web:

```
https://api.github.com/repos/gsi-upm/gsi-upm.github.io?&access_token=XXX

{
  "id": 50830157,
  "name": "gsi-upm.github.io",
  "full_name": "gsi-upm/gsi-upm.github.io",
  "owner": {
    "login": "gsi-upm",
    "id": 2894736,
    "avatar_url": "https://avatars.githubusercontent.com/u/2894736?v=3",
    "gravatar_id": "",
    "url": "https://api.github.com/users/gsi-upm",
    "html_url": "https://github.com/gsi-upm",
     "followers_url": "https://api.github.com/users/gsi-upm/followers",…
```

Para cada petición desde JavaScript se devuelve un array de objetos siendo cada objeto un repositorio con sus propiedades. Es necesario utilizar *callbacks* para procesar la respuesta y extraer la información requerida de cada proyecto.

La lectura de contenidos de GitHub sólo se realiza en la página de administración. En el archivo *admin-functions.js* están definidas todas las funciones relacionadas con el módulo de administración, a excepción de algunas que se encuentran en el archivo *general-functions.js* y que son comunes al módulo de visualización.

En la sección ‘Controles de administrador’ se verán las funciones JavaScript desarrolladas, ya que prácticamente todas utilizan lectura de contenidos de GitHub y escritura en Firebase, por lo que es necesario ver la ‘Escritura de contenidos en Firebase’ antes ver los procedimientos.

3.3.4. ESCRITURA DE CONTENIDOS EN FIREBASE

La base de datos de Firebase guarda la información en formato JSON. Cada nodo se puede entender como una tabla y dentro del mismo, cada elemento como un registro cuyos atributos son los campos. La interfaz visual de Firebase se puede ver en la siguiente figura:

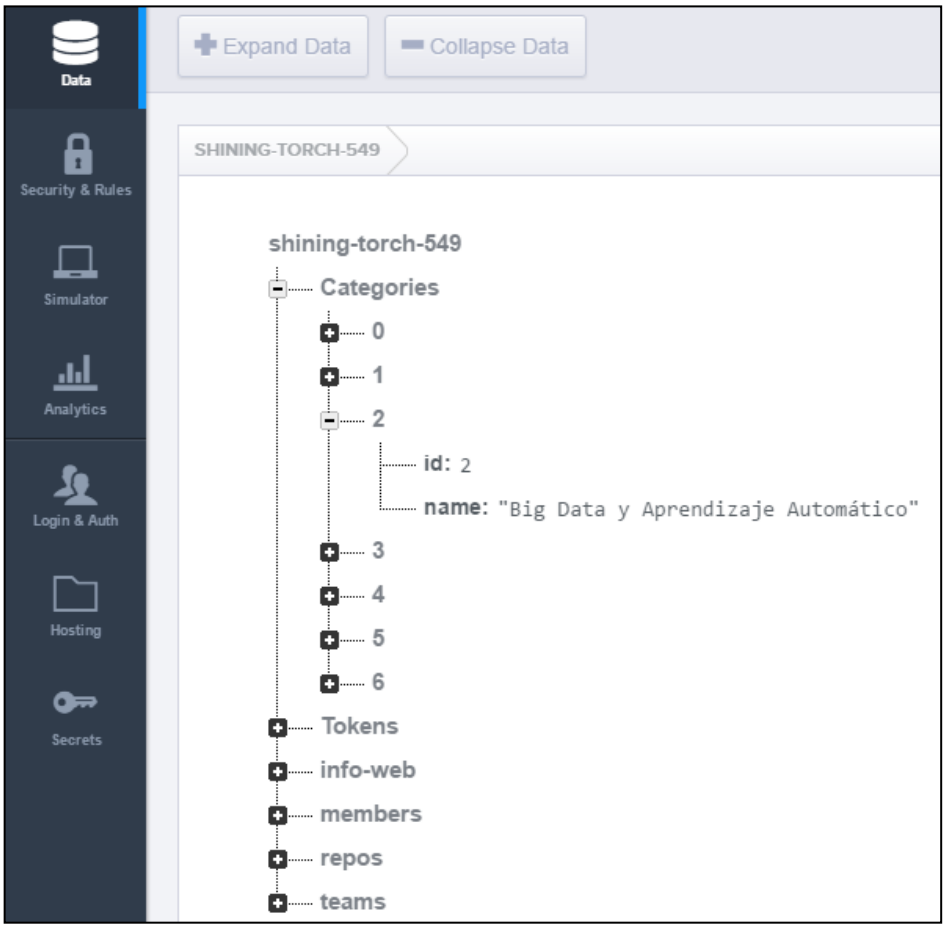

Figura 3.3: Estructura de Firebase

Los nodos *Categories* y *Tokens* son catálogos cuyos datos son invariantes. En el primero se guardan las posibles categorías para los repositorios y en el segundo los *tokens* para GitHub. Los demás nodos guardan la información de equipos, miembros y repositorios respectivamente. En *info-web* se almacenan las fechas de últimas actualización para cada uno de ellos. Cualquier nodo puede ser creado, modificado o eliminado desde el panel de Firebase.

En la sección anterior se indicó cómo se accedía a la información de GitHub, pero no todos los campos de la respuesta son de interés para mostrar en la web. Por ese motivo, al guardar los repositorios, miembros y equipos en Firebase se realiza una selección de campos. El identificador de cada registro no es el nombre del mismo sino su ID de proyecto, ya que el nombre podría variar pero el ID no. En las tablas siguientes se muestran los campos guardados:

Tabla 3.1: Modelo de datos - Miembro

Miembro	
html_url	Url a la página en GitHub de la cuenta
id	Identificador de la cuenta de GitHub
name	Nombre de la cuenta
role	Rol en la organización (miembro o administrador)
url_image	Url a la imagen de perfil de la cuenta

Tabla 3.2: Modelo de datos - Equipo

Equipo	
id	Identificador del equipo de GitHub
members	Array de los miembros pertenecientes al equipo
name	Nombre del equipo

Tabla 3.3: Modelo de datos - Repositorio

Repositorio	
category	Id de la categoría a la que pertenece
codeReadme	Librería para visualizar campo *readme*
collaborators	Colaboradores del repositorio
created_at	Fecha de creación
description	Descripción del repositorio (de GitHub o personalizada)
download_zip_url	Url de descarga de un archivo .zip con el proyecto
html_url	Url a la página en GitHub del repositorio
id	Id del repositorio
language	Lenguaje principal en que está escrito el código del repositorio
name	Nombre del repositorio
owner	Propietario del repositorio
private	Variable booleana que indica si el repositorio en privado en GitHub
readme	Archivo codificado en Base64 del Readme del repositorio
show	Variable booleana que marca si el repositorio es visible en la web
size	Tamaño en KB del repositorio
updated_at	Fecha de última actualización
urlImage	Url a la imagen del repositorio

Los atributos *collaborators* y *readme* se obtiene de dos peticiones distintas. Los campos subrayados son los que no se obtienen directamente de GitHub:

- **description:** si el repositorio posee una descripción en GitHub, al mostrarse en la página de administración de la web ésta aparece y es modificable; si no, se puede introducir una directamente.

- **show:** desde la administración se puede marcar para que el repositorio sea visible o no en la web. Dicho valor se almacena en este campo booleano.
- **urlImage:** se trata de una url a una imagen que se quiera asignar al proyecto.
- **codeReadme:** indica la librería de JavaScript que se debe utilizar para visualizar el archivo Readme del proyecto. Este apartado se verá más detenidamente en el módulo de visualización (Sec. 3.4.3)

Para escribir esta información en Firebase se utilizan las funciones de la librería *Firebase.js*.

3.3.5. GESTIÓN DE CONTENIDOS

Una vez habilitados los controles de administrador tras la identificación, es posible ejecutar un conjunto de acciones que cuyo objetivo es mantener la web actualizada. Todas ellas recuperan información de GitHub y la almacenan en Firebase:

- **Guardar repositorios:** recoge los IDs de los repositorios que tiene en pantalla, pide información a GitHub del mismo y lo guarda en Firebase. Si ya existe el repositorio lo sobrescribe.
- **Comprobar eliminados:** con esta acción se verifica que en Firebase no exista un repositorio que se ha eliminado de GitHub.
- **Actualizar repositorios:** actualiza sólo la información de los repositorios que se obtiene de GitHub y sólo los que ya están en Firebase.
- **Actualizar miembros:** recargar la información de los miembros en Firebase. No guarda ninguna información de la anterior actualización.
- **Actualizar equipos:** mismo procedimiento que los miembros pero para los equipos.

Por motivos de seguridad, si GitHub detecta un token como parte del código de algún repositorio, lo inhabilita y queda inservible. Para evitar esta situación, ya que la web está alojada en GitHub, el token se ha guardado en Firebase y antes de realizar peticiones se asegura de haberlo leído para poder montar la url adecuada.

Las funciones JavaScript desarrolladas del archivo *admin-functions*.js que sostienen estos procesos son:

- ***getTokenFirebase(callback)*:** recoge el *token* asociado a la cuenta d GitHub, almacenado en Firebase. Esta función se ha realizado con callback para asegurarse de las demás funciones esperan a que ésta termina.
- ***loadRepositoriesGithub()*:** carga los primeros nueve repositorios de la respuesta de GitHub en la página de administración.
- ***loadMore()*:** función necesaria para el funcionamiento del 'desplazamiento infinito' (*infinite scroll*). Carga los siguientes nueve repositorios al llegar al final de la pantalla
- ***filterRepositories()*:** función similar a *loadRepositoriesGithub()* aplicando un filtro según el nombre o el id del repositorio.
- ***filterOffAdmin()*:** desactiva el filtro que se pueda establecer con la función anterior
- ***saveRepositories():*** guarda en Firebase la información de los repositorios que se encuentran en pantalla (ya sea aplicando filtro o no). Esta opción guarda tanto la información de GitHub como la configuración aplicada.
- ***checkDeletedRepositories()*:** comprueba si existen repositorios en Firebase que se han eliminado de GitHub. También sirve para verificar si alguno de ellos ha cambiado su nombre.
- ***deleteRepositories ():*** borra los repositorios detectado por la función anterior.
- ***updateRepositories():*** actualiza la información de cada repositorio de Firebase. Sólo se modifica la información que es devuelta por GitHub
- ***updateMembers():*** borra de Firebase toda la información relativa a los miembros y la recarga de nuevo. Realiza dos peticiones: una para los miembros y otra para los

administradores de la organización, ya que el rol de los miembros no es un campo predefinido en la repuesta de GitHub.

- ***updateTeams():*** mismo procedimiento que el anterior pero para los equipos.

En la siguiente figura se muestra un diagrama de secuencia UML de la función *loadRepositoriesGithub()*. Es ejecutada en el callback de *getTokenFirebase(callback), con el token* ya guardado:

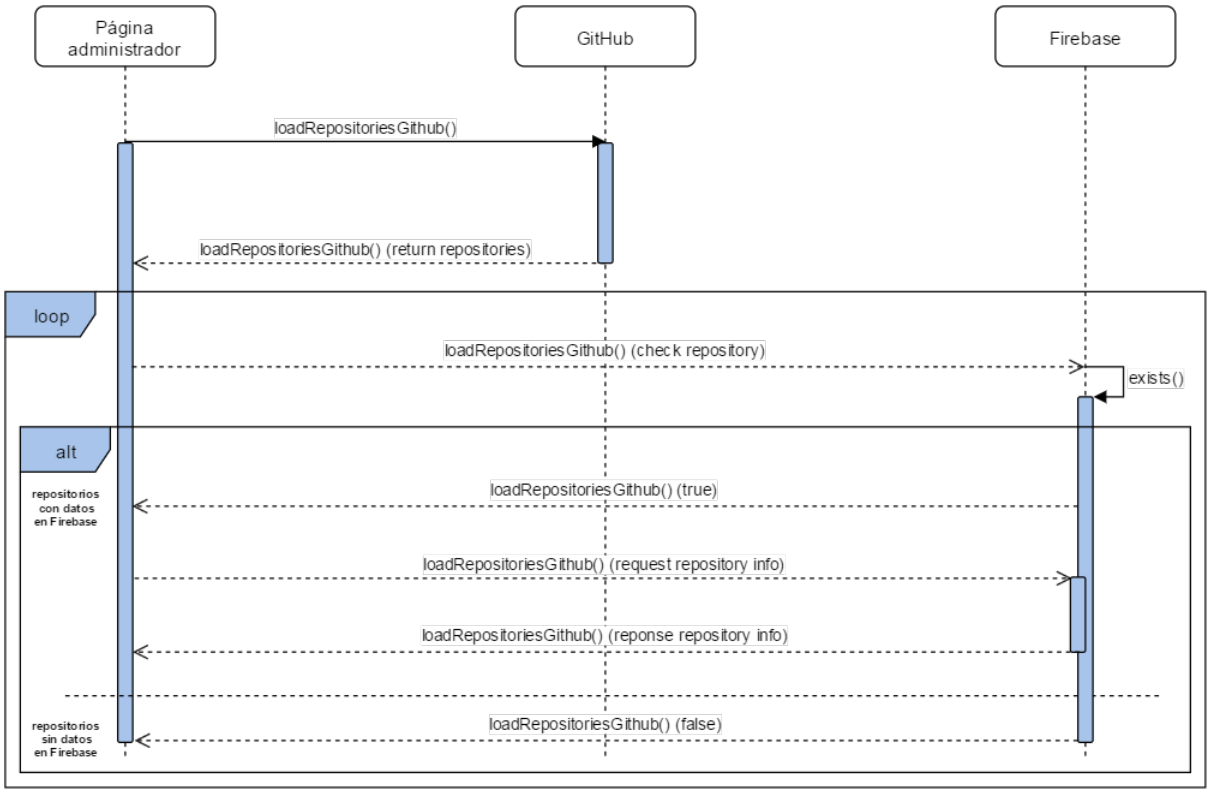

Figura 3.4: Diagrama UML de la carga de repositorios de GitHub

Todas las funciones siguen un esquema similar alterando el orden de ejecución y cambiando la función de lectura de Firebase por la de escritura. No obstante la estructura no varía notablemente.

En la siguiente figura se observa un resumen del Módulo de administración de forma esquemática. Las funcionalidades de actualizar miembros y equipos están disponibles desde cualquier punto de la web en la barra de menú superior, una vez se haya identificado el administrador. Para estos dos casos no existiría el bloque de visualización y la gestión se reduciría a la acción de actualizar.

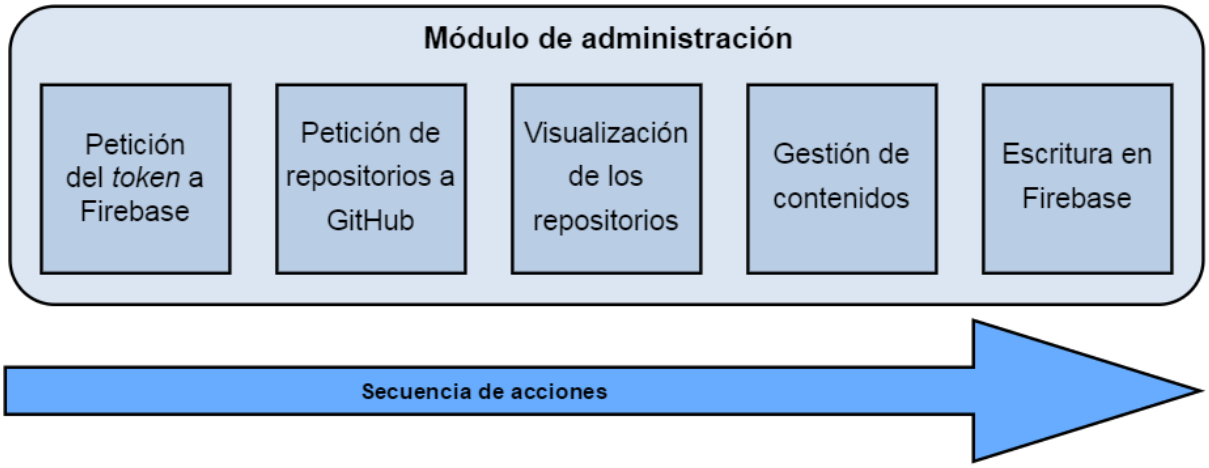

Figura 3.5: Diagrama de flujo del Módulo de Administración

3.4. MÓDULO DE VISUALIZACIÓN

Dentro de este módulo se incluye toda la interfaz pública de la web. El usuario que acceda al portal tendrá a su disposición la información ya guardada en Firebase. Se han desarrollado un conjunto de funciones JavaScript para la visualización de contenidos. Todas ellas se encuentran en el archivo *show-function.js*. El objetivo de este módulo es agregar el contenido a los contenedores HTML para cada página de la web.

Se han utilizado las funciones *appendTo* y *prependTo* de la librería *jQuery.js*. El número de elementos (repositorios, miembros o equipos) a mostrar es variable y éste es el motivo por el que se ha optado por este mecanismo. Además, para muchos campos es necesario maquetar la información devuelta por Firebase usando JavaScript.

Para analizar la arquitectura de este módulo se van a ver las páginas por separado con las funciones que las sustentan.

3.4.1. PÁGINA GLOBAL DE REPOSITORIOS

En esta página se puede visualizar de forma resumida todos los repositorios guardados y realizar búsquedas con diferentes filtros sobre los mismos. Al cargar la página se ejecuta la función *showAllRepos()* y al realizar filtros se ejecuta *filterRepositoriesAll()*. A continuación se detalla cada una:

- ***showAllRepos()*:** lee todo el contenido del nodo 'repos' de Firebase, recorre la respuesta y guarda en memoria todos los repositorios en una variable global, *reposSortGlobal*. Comprueba cuáles se deben mostrar y los añade a un *array*. La maquetación de los campos se explicará en la sección 3.4.3, 'Página individual de repositorio'. Con los datos en memoria, no se vuelve a realizar la petición a Firebase hasta que no se pulsa el botón 'Vaciar', ya que utiliza esta misma función para eliminar todos los filtros. De esta manera se consigue que las búsquedas se ejecuten a mayor velocidad ya que sólo tienen que recorrer un *array* en memoria. Esta fue la medida que se tomó debido a que no se pueden utilizar de una manera simple las sentencias de consultas a bases de datos típicas, como *ORDER BY* o *WHERE*.

- ***filterRepositoriesAll()*:** en el lateral de la página se ha desarrollado un panel con diferentes filtros y criterios de ordenación. Esta función comprueba cuáles están activos, qué repositorios cumplen y muestra los coincidentes. Al estar el conjunto de repositorios en memoria, sólo tiene que recorrer cada uno y aplicar los métodos *indexOf*, *substring*, *toUpperCase* y *replace* de JavaScript. En la siguiente sección se verá un mecanismo que facilita Firebase para poder realizar una consulta filtrada, pero se trata de un filtro de coincidencia exacta y no es lo que se buscaba para esta página.

- ***filterOffAll()*:** vacía los criterios de búsqueda y devuelve la ordenación a su posición inicial, ordenados por fecha de actualización en orden descendente. Llama a la primera función que recarga todo el contenido.

En la siguiente figura se muestra un diagrama UML de *showAllRepos()*. La función del filtro no interactúa con ningún servidor, luego toda su acción se realiza en el lado del cliente.

Figura 3.6: Diagrama UML de la función showAllRepos()

3.4.2. PÁGINAS DE REPOSITORIOS POR CATEGORÍAS

Al igual que en la página anterior, en ésta se muestran de forma resumida los repositorios pertenecientes a una categoría concreta. Uno de los objetivos principales de la web era la categorización de los proyectos. Se han tomado las categorías ya existentes en la web oficial del GSI [8]. En la sección 3.3.4 se mencionó la creación de dos catálogos en Firebase bajo los nodos *Tokens* y *Categories*. En el segundo nodo se han almacenado las posibles categorías que pueden ser asignadas y un identificador numérico a cada una. Esta decisión se tomó con la previsión de modificaciones en las mismas o la inclusión de alguna adicional. Actualmente existen siete:

Tabla 3.4:Catálogo de categorías

Categoría	Id
Sin asignar	0
Agentes y Simulación Social	1
Big Data y Aprendizaje Automático	2
NLP y Análisis de Sentimientos	3
La Web de Datos y Tecnologías Semánticas	4
Ingeniería Web y de servicios	5
Otros	6

La primera categoría sólo se utiliza a nivel funcional, para marcar aquellos repositorios guardados en Firebase pero sin categoría asignada. Para cada categoría se ha creado una página en la web y mediante una misma función pasando parámetros se consiguen mostrar los repositorios pertenecientes:

- ***showRepositoriesByCategory(categoryParam, idShowDiv)*:** recibe como parámetros de entrada el parámetro de la categoría y el ID del contenedor donde se deben anidar los repositorios. El primer parámetro es una cadena de texto con el nombre de la categoría, de la que es necesaria obtener el ID del catálogo de categorías. Para cada proyecto

almacenado en Firebase el campo *category* contiene el ID de la misma, luego se necesita el id de la categoría que llama a esta función.

En la siguiente figura se muestra el diagrama UML de esta función. Realiza dos llamadas a Firebase; una para conseguir el id de la categoría y otra para pedir los repositorios de la categoría:

Figura 3.7: Diagrama UML de la función showRepositoriesByCategory()()

El código HTML de las páginas de categoría mantiene una estructura uniforme. A continuación se muestra el de una de ellas, 'Agentes y Simulación Social' (CategoriaASS.html):

```
<div class="page-header" style="margin-top:10px;" align="center">
    <h1 id="title-WDT">{{page.title}}</h1>
</div>
<div id="container-WDT">
       <div class="row">
              <div class="col-md-6" id="column-WDT-1"></div>
              <div class="col-md-6" id="column-WDT-2"></div>
       </div>
</div>
<script type="text/javascript">
  $(window).load(function(){
    showRepositoriesByCategory(document.getElementById('title-
WDT').innerHTML,'WDT');
  });
</script>
```

Se utiliza Liquid para acceder al título de la página, definido en el Front Matter. El contenido se divide en dos columnas de la hoja de estilos proporcionada por Bootstrap. La función se ejecuta al cargar la página y se observa que los parámetros pasados son el nombre de la categoría y el identificador de las columnas donde se mostrarán los repositorios. Mediante una lógica interna de la función se colocan los repositorios en dos columnas: en cada paso del bucle de repositorios una variable toma el valor 1 ó 2 y con ella se forma el identificador de la columna.

La web se ha diseñado para poder añadir categorías o modificarlas sin tener que cambiar la estructura completa de la web. Al elegir la categoría en la administración, las opciones disponibles se rellenan de forma dinámica con el catálogo de Firebase, luego para modificar el nombre de una de ellas habría que realizar los siguientes pasos:

1. Modificar en el catálogo de Firebase el nombre de la categoría.
2. Modificar en el Front Matter la página correspondiente el nombre de la categoría.

Para crear una nueva serían necesarios los mismos pasos pero creando la nueva categoría en Firebase y la nueva página con el mismo esquema anterior. En la sección 3.5 se tratará la estructura seguida en cuanto a la paginación y menú de la web ya que se utilizó el Front Matter para organizar las páginas desarrolladas.

3.4.3. PÁGINA INDIVIDUAL DE REPOSITORIO

En esta página se puede ver la información almacenada en Firebase de un repositorio en concreto. Se puede acceder mediante cualquiera de las dos páginas analizadas anteriormente pulsando encima del contenedor de uno de los repositorios.

Se ha diseñado una página Repositorio.html a la que se le debe pasar el ID del repositorio como parámetro en la URL. Otra forma habitual de pasar una variable entre páginas hubiera sido usando PHP, pero ya se explicó que en sitios webs alojados en GitHub no está permitido su uso. Para añadir el ID del repositorio se ha usado la siguiente función que es asignada al control del evento *onclick* de cada contenedor que muestra la información resumida:

- ***addIdReposToURL(idRepo)***: su funcionamiento es simple: añade a la URL de la página individual de cada proyecto el ID del repositorio que se pasa por parámetro y redirige a la misma:

```
function addIdReposToURL(idRepo){
    reposUrl = "/Repositorio.html?";
    reposUrl = reposUrl + "&id=" + idRepo;
    window.location.href = reposUrl;
}
```

En la página Repositorio.html, una vez cargada, se recoge la URL y se busca el parámetro pasado, comprueba que sea numérico y llama a la función desarrollada que muestra el contenido individual. Si el parámetro no es numérico o no existe se redirige a la página global de repositorios:

- ***showInfoRepository(idParam)***: recibe el ID obtenido de la URL y comprueba si existe. En caso afirmativo maqueta los campos y monta la disposición de los mismos en varias columnas para añadirlos a un contenedor del HTML. En caso de que el repositorio exista pero la propiedad *show*, la que marca si debe ser público, indique que no lo es, lo notifica.

Para la mayoría de los campos no es necesario mucho tratamiento, tan sólo cambiar la ordenación de la cadena de texto en las fechas, controlar si el tamaño excede los KB para transformarlo a MB o situar todos los colaboradores del repositorio seguidos. Sin embargo, para una correcta visualización del archivo Readme sí es necesario utilizar funciones auxiliares.

Ya se mencionó que el resultado de la petición del archivo Readme a GitHub era el propio contenido codificado en Base64. El contenido es guardado en Firebase sin modificar. Además, se diseñó una lógica para analizar la extensión del mismo archivo y así guardar un campo que indicase el método de transformación a HTML que se debe seguir. Existen tres grupos de posibles extensiones que se deben tratar de forma separada:

1. **.rdoc**: estos archivos han sido creados usando RDoc, una herramienta para generar documentación basada en el lenguaje de marcado SimpleMarkup para proyectos en Ruby. Una vez decodificado de Base64 queda el código en el lenguaje de marcado pero el objetivo es convertirlo a HTML. Para ello se ha utilizado la librería Rundown, ya explicada (Secc.2.5.5). Si el archivo Readme tiene esta extensión, en el campo *codeReadme* se guarda ‘rundown’.

2. **.txt**: para el contenido de los archivos con esta extensión no es necesaria ninguna librería especial, sólo añadir la etiqueta *pre* de HTML al resultado de la decodificación de Base64. Esta etiqueta marca que el texto incluido es texto plano. El campo *codeReadme* toma el valor 'txt' en este caso.

3. **Resto de las extensiones**: para las extensiones Markdown o los archivos sin extensiones se utiliza la librería Showdown (Secc.2.5.5). Realiza la misma función que Rundown pero para archivos escrito en el lenguaje de marcado de Markdown. Todos los archivos Readme que no tengan ninguna de las dos extensiones anteriores utilizan este mecanismo, es decir, *codeReadme* toma el valor de 'showdown'.

A continuación se muestra el fragmento de código de la función ***showInfoRepository()*** que decodifica el archivo Readme según lo expuesto:

```
switch(infoRepo.codeReadme) {
  case "showdown":
    var converter = new showdown.Converter();
    var decodedReadme = converter.makeHtml(decodeBase64(infoRepo.readme));
    break;
  case "rundown":
    var converterRDoc = new Attacklab.rundown.converter();
    var decodedReadme = converterRDoc.makeHtml(decodeBase64(infoRepo.readme));
    break;
  case "txt":
    var decodedReadme = "<pre>" + decodeBase64(infoRepo.readme) + "</pre>"
    break;
  default:
    var decodedReadme = "<h2>Repositorio sin archivo Readme<h2>";
}
```

La función ***decodeBase64(string)*** se ha desarrollado para decodificar de Base64 usando la función *decodeURIComponent()* de JavaScript. La siguiente figura contiene fragmentos de los resultados de los dos pasos de decodificación necesarios para un archivo Readme en Markdown:

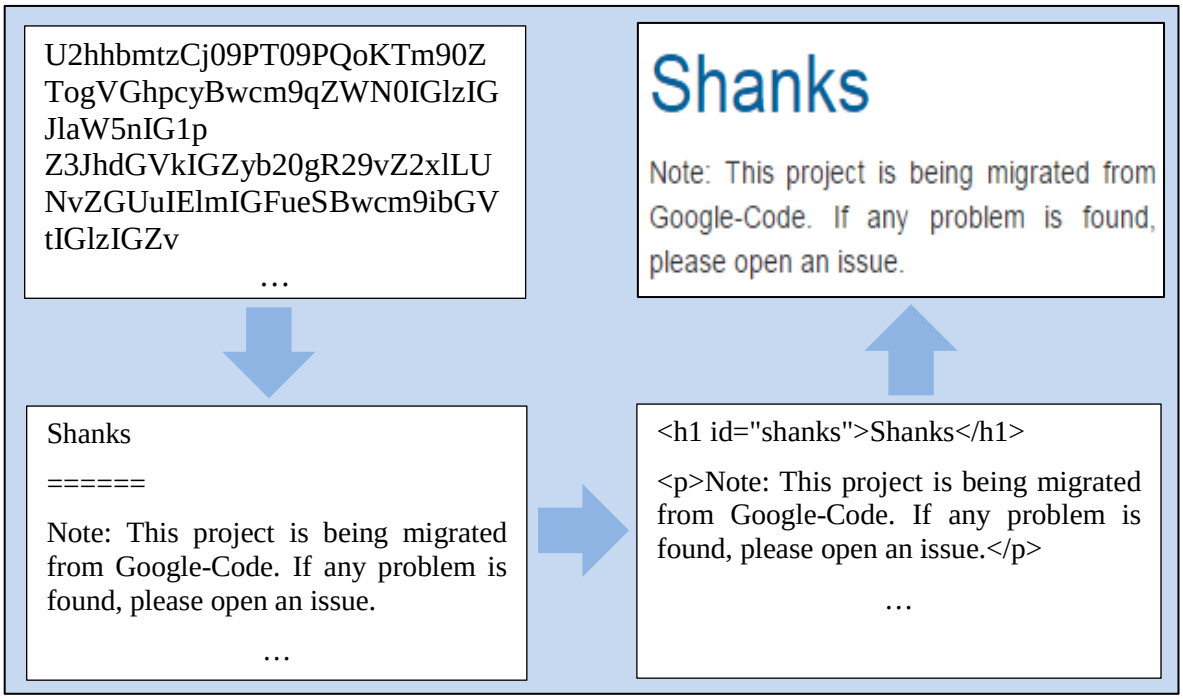

Figura 3.8: Proceso de decodificación del archivo Readme

Con este sistema cualquier modificación realizada desde GitHub se mostrará en esta página cada vez que se actualicen los repositorios.

3.4.4. PÁGINA DE MIEMBROS

En la página de miembros, Miembros.html, se muestran todos los colaboradores que pertenecen a la organización de GitHub del GSI. El contenido sobre los miembros se encuentra almacenado en el nodo *members* de Firebase. El identificador de cada registro con la información del miembro es el propio ID de la cuenta de GitHub del mismo.

Al igual que con las otras páginas, se ha desarrollado una función que asigna a un contenedor los datos de los miembros dispuestos en seis columnas. Como ya se mencionó, existen dos roles posibles: miembro y administrador. A continuación se explica su funcionamiento:

- ***showMembers()***: realiza una petición a Firebase para obtener el conjunto de los miembros que se almacenan en un array de objetos cuyos atributos son los campos. Para cada miembro comprueba si es administrador o no. Y en caso afirmativo incluye una imagen de una estrella para marcar la posición privilegiada. Se utiliza la url del avatar de cada cuenta como hipervínculo a la página oficial de GitHub del miembro.

3.4.5. PÁGINA DE EQUIPOS

La estructura de esta página es muy similar a la anterior. El mecanismo de acceso al contenido de los equipos es idéntico, accediendo al nodo *teams* en este caso. Como se ve en la siguiente figura, los miembros pertenecientes a cada equipo vienen dentro de un nodo hijo., por lo que es necesario añadirlos a un array independiente:

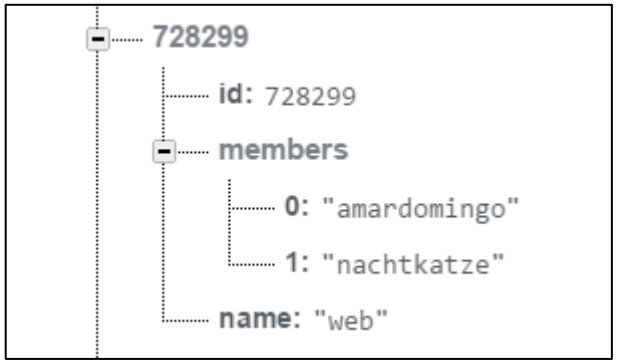

Figura 3.9: Árbol del nodo *teams*

La función desarrollada para esta página es:

- ***showTeams()***: realiza una petición al nodo *teams* de Firebase y recorre la respuesta. Para cada equipo monta el contenido dentro de un contenedor desplegable perteneciente a Bootstrap y adaptado al estilo de la web.

3.5. FUNCIONAMIENTO GENERAL

En este apartado se recogen los aspectos comunes a ambos módulos, como son la estructura de carpetas del proyecto, la utilización del Front Matter y Liquid o el archivo de configuración de Jekyll. Todos ellos son características propias del generador de sitios estáticos utilizado.

3.5.1. FRONT MATTER EN LA WEB

En la sección 2.1.4 se explicó qué era el Front Matter y su utilidad. Todas las páginas de la web realizada con Jekyll tienen uno. Se han definido las siguientes variables:

- **layout**: todas ellas utilizan el layout *default*. En él se incluye el menú superior, el pie de página y el contenido de la página se muestra en la parte central, en el objeto *container* de Bootstrap. En el siguiente capítulo se mostrará de forma gráfica el contenido del mismo.
- **title**: es el título de la página y el que se muestra en el navegador. Se accede al mismo utilizando Liquid como ya se explicó.
- **group**: las páginas que se deben mostrar en el menú superior de navegación de la web tienen esta variable con el valor *navigation*.
- **description**: esta variable se utiliza de manera conjunta con al anterior y sólo la tienen las páginas de las categorías, ya que éstas no se asignan directamente al menú, sino que se agrupan bajo el título Categorías. Toma el valor *categories*.
- **order**: todas las páginas que aparecen en el menú tienen esta variable de forma que se ordenan siguiendo este criterio. Toma un valor numérico indicando la posición que debe ocupar en el menú.

Se ha desarrollado un archivo escrito en Liquid para generar de forma dinámica el menú de las páginas. Es llamado de la siguiente manera desde el layout *default* ya que el menú es común a todas las páginas:

```
<ul class="nav navbar-nav">
       {% assign pages_list = site.pages %}
       {% assign group = 'navigation' %}
       {% include JB/pages_list %}
</ul>
```

La lógica de este fichero es buscar las páginas de la web que tienen la variable *group* igual a *navigation* y asignarlas al menú. Si además tienen la variable *description* igual a *categories* las añade a un menú desplegable. El siguiente fragmento de código corresponde a este archivo y su función es escribir el HTML necesario para mostrar los elementos de la lista del menú, para los que no son categorías, y seleccionar la página activa:

```
{% if page.url == node.url %}
  <li class="active">
  <a href="{{ BASE_PATH }}{{node.url}}" class="active">{{node.title}}
  </a></li>
{% elsif node.title == "" %}
{% else %}
  <li>
  <a href="{{ BASE_PATH }}{{node.url}}">{{node.title}}
  </a></li>
{% endif %}
```

3.5.2. ESTRUCTURA DE CARPETAS DE LA WEB

Las carpetas necesarias para generar el sitio web son las siguientes:

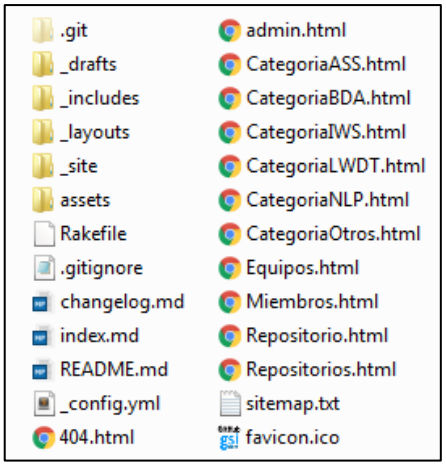

Figura 3.10: Carpetas antes de generar la web

La estructura de carpetas de Jekyll se analizó en la sección 2.2.3. Todas las librerías de JavaScript y las hojas de estilos se encuentran en la carpeta *assets*, en el tema de Bootstrap. Una vez que Jekyll ha generado el sitio web a partir de estos directorios, se crea la carpeta _site con el siguiente contenido:

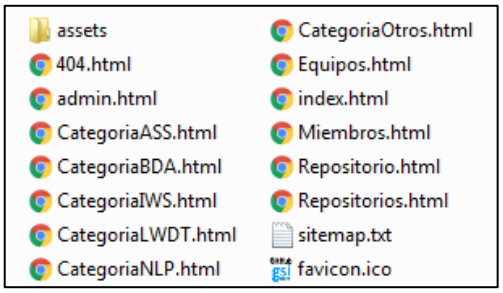

Figura 3.11: Carpetas después de generar la web

Este es el contenido que se crea cuando se ejecuta la web en local, antes de subirla a GitHub. La estructura final es muy similar a la de cualquier página web. El propio repositorio en GitHub en el que se aloja tiene todos los archivos de la Figura 3.10, y es cuando es llamado cuando se genera la web y se obtiene la segunda estructuración de carpetas.

3.5.3. BOOTSTRAP EN LA WEB

Para el diseño de la web se partió desde una plantilla básica que se genera al instalar Jekyll Bootstrap. Esta plantilla estaba orientada hacía un blog por lo que no se pudo reutilizar casi ningún contenido de la misma. Sin embargo, sí tuvo utilidad para ver los elementos de Bootstrap que utilizaba y que hacían que la web fuese adaptativa (*responsive*).

Las clases de la hoja de estilos proporcionada por Bootstrap que más se han utilizado son:

- *jumbotron*: para todos los contenedores (repositorios, miembros y equipos)
- *row*, *col-md-x*: para la distribución en columnas de los contenedores
- modal: para los mensajes de información, avisos o la identificación de usuario. Se ha utilizado además su estructura: modal-content, modal-header, modal-body, modal-footer…
- *form-group*: para los formularios de introducir datos junto con los *inputs* y los controles de botones.
- *collapse*: para los contenedores que se contraen y se expanden mostrando contenido
- *hidden-xs*: se ha utilizado esta clase para que las notificaciones que aparecen en pantalla tras realizar una acción no se muestren en la versión móvil por problemas de

compatibilidad. Todo el contenido incluido en el contenedor que posea esta clase no se mostrará en móviles ni tablets.
- *dropdown-menu*: para los combos desplegables.
- *navbar*: esta clase se ha utilizado para el menú principal de la web y para la barra de control de la página de administrador.

Además de estas clases se han generado muchas otras en una hoja de estilos, *style.css*, con todo el CSS personalizado.

De manera adicional se ha diseñado un efecto de 'scroll infinito' en la página de administración. Esto afecta tanto a nivel visual como funcional. Ya se explicó la función que cargaba los repositorios en pantalla en esta página, *loadRepositoriesGithub()*. Lo que faltaba por explicar es que esta función sólo carga nueve repositorios cada vez que es llamada, es decir, sólo pide los siguientes nueve repositorios a GitHub. Mediante un manejador se controla cuándo el scroll llega al final de la página y vuelve a la llamar a la función que carga los nueve siguientes. El objetivo de este funcionamiento es agilizar la carga de la página y evitar que se tengan que cargar todos los repositorios al mismo tiempo en una misma petición.

En la página de Repositorios, donde se visualizan todos y se pueden realizar búsquedas, no se ha implementado esta funcionalidad. El motivo es que esta página carga todos los repositorios de Firebase al principio, los guarda en memoria como ya se explicó y luego realiza todos los filtros sobre ese conjunto. En este contexto no tenía mucho sentido realizar una carga según scroll, ya que no se ahorraban recursos al estar todos ya en memoria.

3.5.4. PIE DE PÁGINA

Al igual que el menú, el pie de página de la web se mantiene constante en toda la web. Se han utilizado variables del archivo de configuración de Jekyll para mostrar información sobre la página. Para acceder a ellas se ha utilizado Liquid:

```
<p>
        Copyright  &copy;
        <a href="http://www.gsi.dit.upm.es/es/" target="_blank">
                {{ site.author.name }}
        </a>
        , {{ site.author.university }} {{ site.time | date: '%Y' }}
</p>
```

Se ha situado además en este apartado los datos de contacto del GSI como son su canal de YouTube, Twitter, GitHub y localización geográfica.

3.6. RESUMEN

En este capítulo se ha analizado toda la funcionalidad y estructura de la web. Desde la petición de información a GitHub hasta la visualización de los repositorios. El concepto principal que marca todas las decisiones es la ausencia de *back-end* de la página. Firebase toma el rol de base de datos pero toda la interconexión se realiza en JavaScript, desde el lado del cliente. El resultado final y la apariencia visual se dejan para el siguiente capítulo, 'Caso de estudio'.

4. CASO DE ESTUDIO

4.1. INTRODUCCIÓN

En este capítulo se muestra el resultado visual final de la web y su manejo por parte del administrador. En primer lugar se analiza la versión de escritorio y finalmente su visualización desde dispositivos móviles. La web se ha diseñado para ser compatible con todos los navegadores en su versión más reciente, ya que se han utilizado componentes HTML5 y CSS3. Las imágenes que se muestran se corresponden al navegador Google Chrome.

4.2. PÁGINA PRINCIPAL

La página inicial de la web se corresponde con el archivo **index.html**.

Figura 4.1: Página principal

La barra de menú superior está fija respecto el resto del documento, de manera que siempre ocupa la misma posición al desplazarse. Para ello se ha utilizado la clase de Bootstrap *navbar-fixed-top*. La sección Categorías es un menú desplegable formado con las páginas de las categorías. En la Sección 3.5.1 del capítulo anterior se detalla la creación del mismo. Al moverse entre las páginas, queda reflejado en el menú la página activa en ese momento.

Figura 4.2: Menú global

Cada uno de los bloques de las categorías en la página principal tiene asociado un contenedor con la clase *collapse* de Bootstrap y que se muestra al realizar click en él. Si se pulsa el botón 'Ver más' la web redirige a la página individual de la categoría pulsada.

Figura 4.3: Bloque de categoría desplegado

La parte inferior de la página principal está compuesta por enlaces a las demás páginas de la web y por el pie de página (*footer*).

Figura 4.4: Parte inferior de la página principal

4.3.PÁGINA DE LOS MIEMBROS

Esta sección de la web se corresponde con la página **Miembros.html**. Los miembros de la organización del GSI de GitHub se sitúan en columnas de forma dinámica. Se puede observar, como ya se explicó en la arquitectura de la web, que los miembros administradores de la organización tienen un icono con una estrella justo al lado de su nombre.

Figura 4.5: Página de los miembros

En la parte superior se muestra además el número de miembros y la última vez que se actualizaron.

4.4.PÁGINA DE LOS EQUIPOS

El archivo **Equipos.html** es al que se llama al entrar en este apartado. Su estructura es similar a la anterior página. Cada equipo contiene otro contenedor desplegable en el que se enumeran los miembros que pertenecen a cada uno. La fecha en que se actualizaron por última vez también se encuentra en la parte superior.

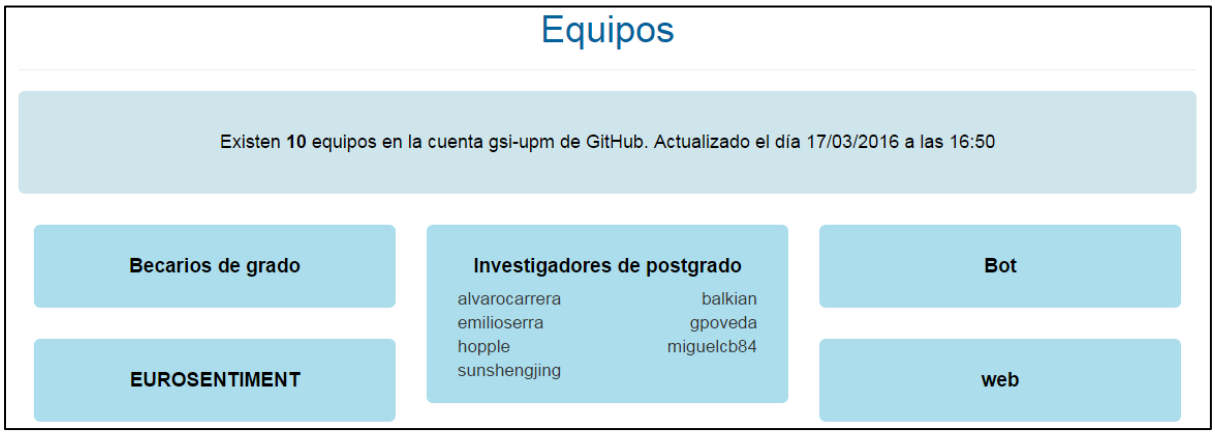

Figura 4.6 Página de equipos

4.5. PÁGINA DE CATEGORÍA

Cada categoría tiene su propia página donde se muestran los repositorios pertenecientes a la misma situados en dos columnas. Al realizar click en uno de ellos la web redirige a su página individual. Los archivos con esta estructura son: **CategoriaASS.html**, **CategoriaBDA.html**, **CategoriaIWS.html**, **CategoriaLWDT.html**, **CategoriaNLP.html** y **CategoriaOtros.html.**

Figura 4.7: Página de categoría

4.6. PÁGINA DE REPOSITORIOS

El módulo de visualización de repositorios se presenta al llamar al archivo **Repositorios.html**. En esta sección es donde se pueden realizar búsquedas según nombre, id o autor de los proyectos. Además se puede buscar por fecha de creación o de actualización, ya sea una fecha concreta o un rango, como un año o un mes en concreto. La ordenación es personalizable según los criterios de nombre, fecha de actualización y fecha de creación, en sentido ascendente y descendente. Búsqueda y ordenación son compaginables.

Para la selección de fechas se han utilizado las librerías *bootstrap-datetimepicker.min.js*, *moment-with-locales.min.js* y *es.js*. La primera facilita un objeto que despliega un calendario al ser pulsado. Las otras dos librerías sirven para cambiar el idioma al castellano y cambiar el formato de la fecha al buscado.

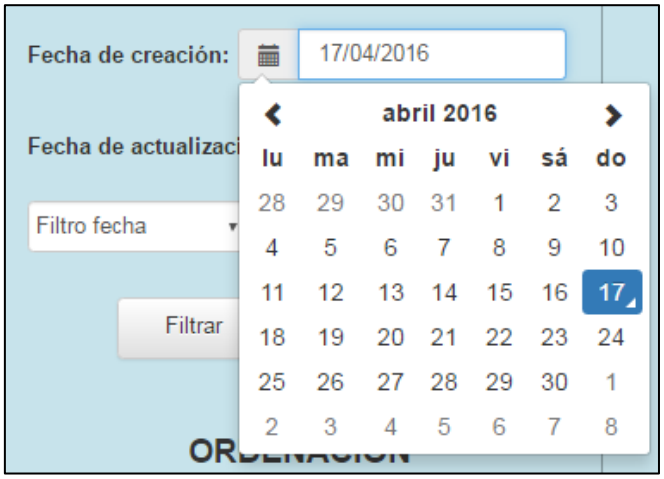

Figura 4.8: Selector de fecha Bootstrap

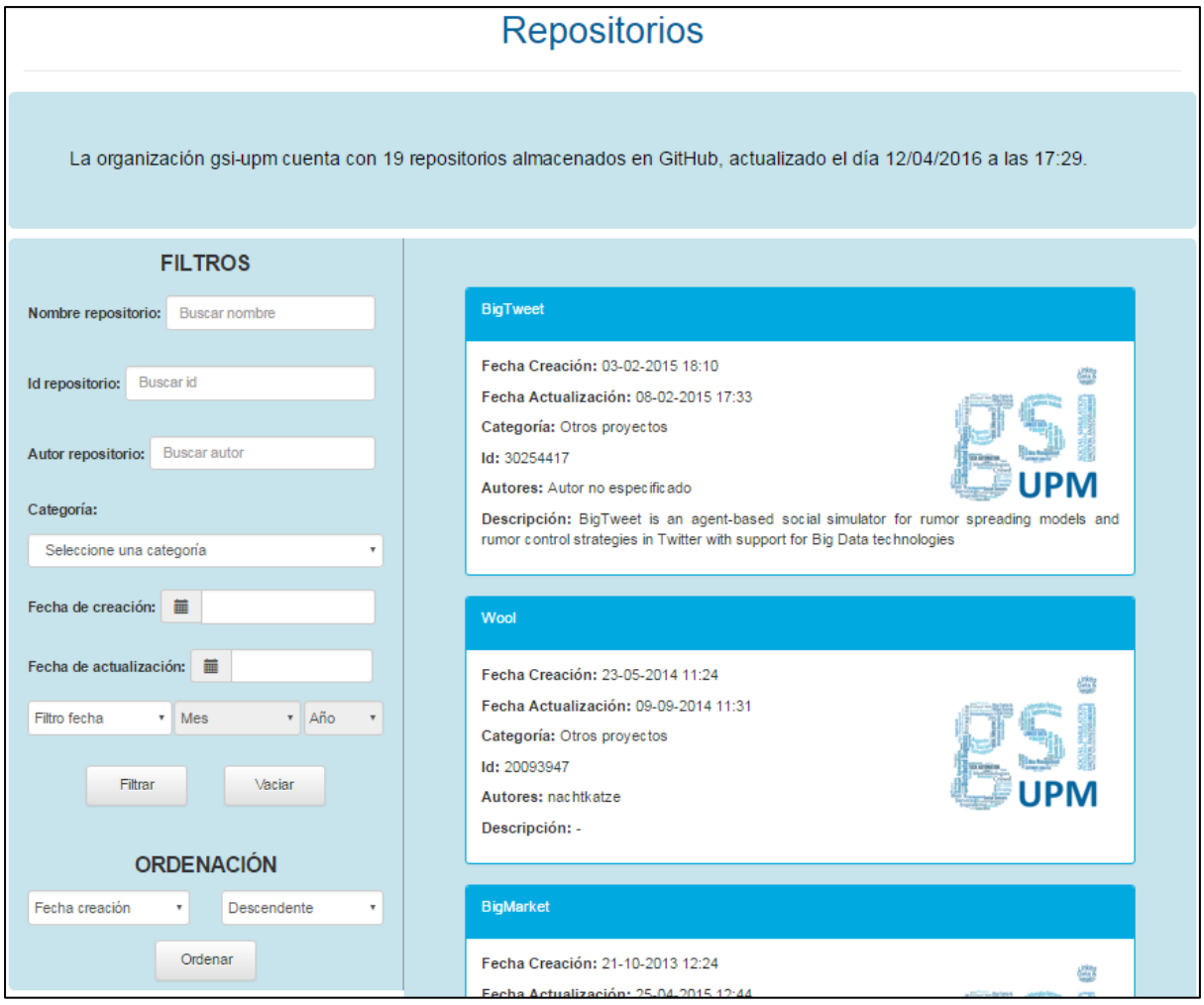

Figura 4.9: Página de repositorios

El bloque de filtros y ordenación se mantiene fijo al desplazarse por la página con el objetivo de facilitar las búsquedas. Para ello se ha creado una clase *.affix* que asigna una posición fija y permite mostrar contenido por encima del contenedor contiguo. Si se realiza click encima de un repositorio la web redirige a la página individual del repositorio.

Si el dispositivo en que se visualiza la web tiene un ancho demasiado pequeño como para anclar la barra de búsqueda, ésta se sitúa en la parte superior y el listado de repositorios a continuación. Esto se ha logrado usando *Media Queries* en el CSS. Estas reglas se han utilizado para más elementos de la web, como en el pie de página para situar correctamente los datos de contacto o las cajas de texto para introducir datos en los filtros.

4.7. PÁGINA INDIVIDUAL DE REPOSITORIO

Cada repositorio tiene su página individual creada de forma dinámica pasando el id del mismo como parámetro en la URL hacia **Repositorio.html** (Sec. 3.4.3). La página está alojada en GitHub, en la cuenta del GSI como un repositorio más. Las capturas que se presentan a continuación son las de este repositorio.

En esta sección se encuentra toda la información almacenada para un repositorio. En la parte superior se ha situado un botón para volver a la página anterior, que puede ser la página de la categoría o la global con todos los proyectos.

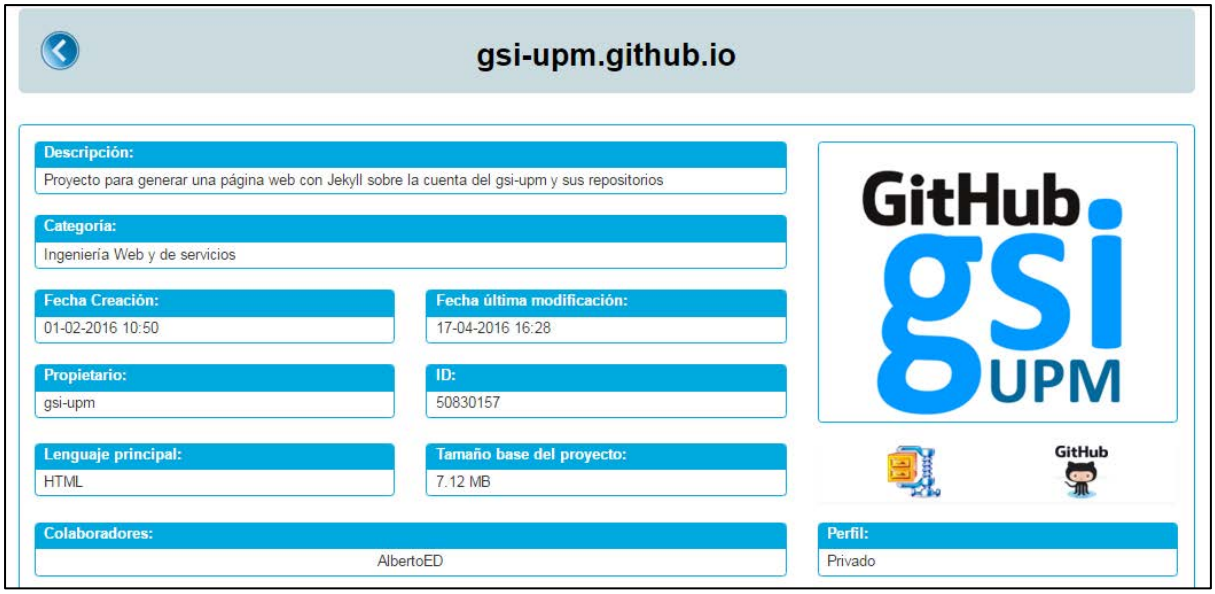

Figura 4.10: Página individual de repositorio

El icono Win-Zip descarga el repositorio comprimido y el de GitHub abre una pestaña con la página oficial del repositorio en GitHub.

El archive Readme descodificado se sitúa justo debajo. Las dos siguientes figuras exponen el mismo archivo: el primero en la página de GitHub del repositorio donde se escribió usando Markdown y el segundo lo que la página web muestra. Se puede observar que son muy similares y que el proceso seguido para su obtención funciona correctamente.

Figura 4.11: Archivo Readme en GitHub

Figura 4.12: Archivo Readme en la web desarrollada

4.8. VENTANA DE IDENTIFICACIÓN

En la parte derecha del menú global de la web se encuentra un botón *Login* que abre una ventana para introducir las credenciales de administrador. Si se introducen correctamente salta una notificación de éxito y redirige a la página de administración, la cual no es accesible por ningún otro método.

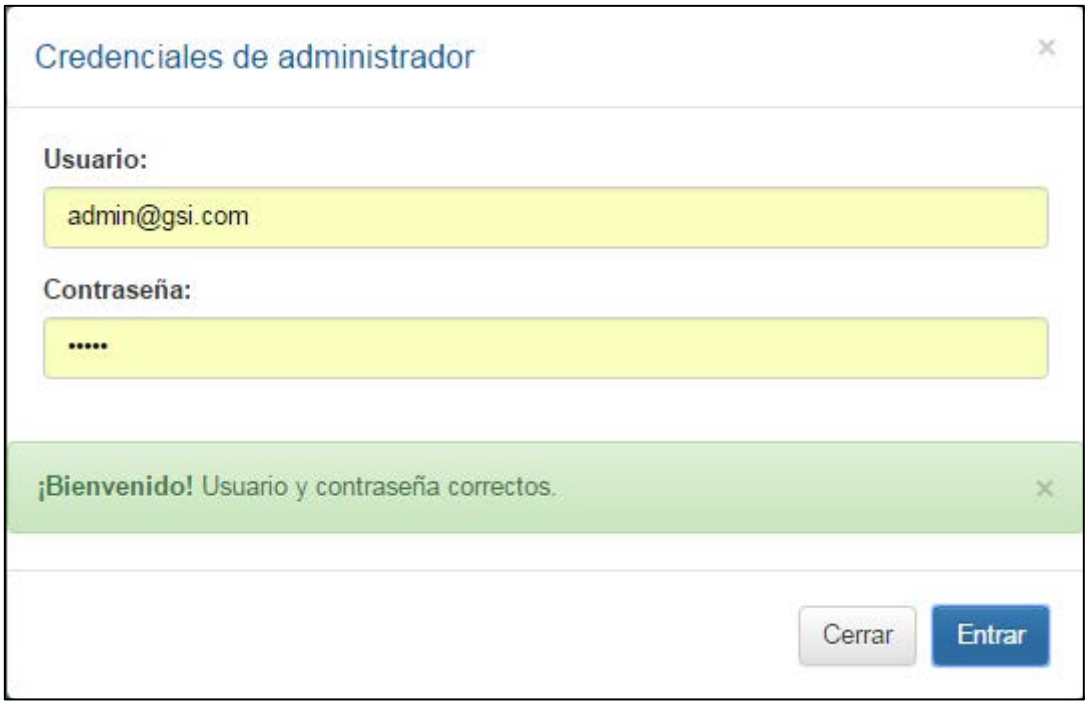

Figura 4.13: Ventana Login - Usuario correcto

Si el usuario no es correcto, se muestra la siguiente notificación:

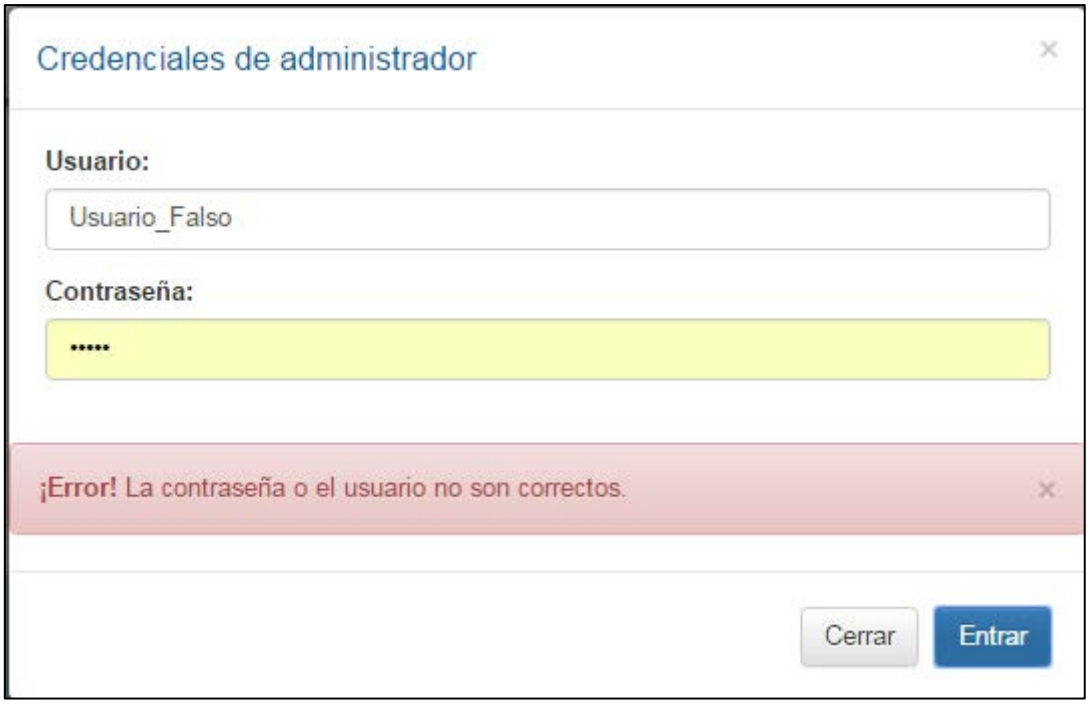

Figura 4.14: Ventana Login - Usuario incorrecto

4.9. PÁGINA DE ADMINISTRACIÓN

En esta página se realiza toda la configuración de los repositorios. Se corresponde con el archivo **admin.html**. Una vez identificado el administrador, la barra de menú superior se modifica para ocultar el botón *Login* y mostrar los elementos de administrador mostrados en la siguiente figura.

Figura 4.15: Menú de administrador

Del menú desplegable, Miembros, Equipos e Información son ventanas emergentes de la clase *modal* de Bootstrap, similares a la del *Login*. Repositorios es el acceso a la página de administrador, ya que esta no aparece junto con los demás elementos del menú.

Desde las ventanas Miembros y Equipos se ejecutan las funciones de actualizar miembros y equipos respectivamente. La ventana de Información es simplemente una pequeña guía dirigida al administrador sobre el mantenimiento de la web.

El nombre del usuario se muestra junto al botón Administrar. Actualmente sólo hay una cuenta de administrador pero existe la posibilidad de crear más en Firebase, si se considera necesario.

Figura 4.16: Página de administración

En la figura 4.16 se puede ver la interfaz de la gestión de repositorios. Los tres botones se corresponden con las tres funciones que se detallaron en la sección 3.3.5. Todos ellos al realizar click abren una ventana para confirmar la decisión. En el caso de Eliminados, en la ventana además aparecen los repositorios detectados que ya no existen en GitHub y permite eliminarlos de Firebase.

Figura 4.17: Ventana de eliminación de repositorios

De manera individual, en cada proyecto se ha dispuesto un *toggle* de la librería *bootstrap-toggle.min.js* para seleccionar si se debe mostrar. La selección de categoría se realiza en este paso. La URL de una imagen es opcional. Por último se puede escribir una descripción personalizada o dejar la que tenga por defecto de GitHub. Este también es un campo opcional.

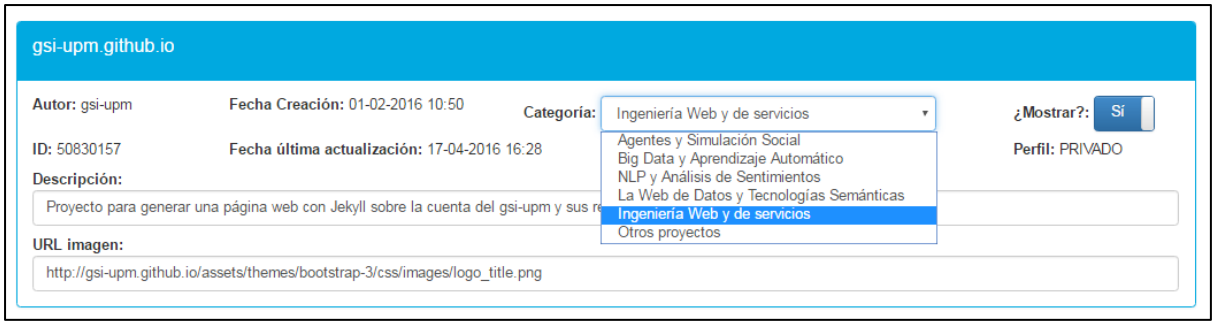

Figura 4.18: Página de administración - Repositorio

4.10. PÁGINA NO ENCONTRADA

Además de las páginas con contenidos, se ha diseñado una página especial con el error 404, no encontrado, para cualquier ruta no existente a la que se trate de acceder. Se ha incluido un link a la página principal de la web.

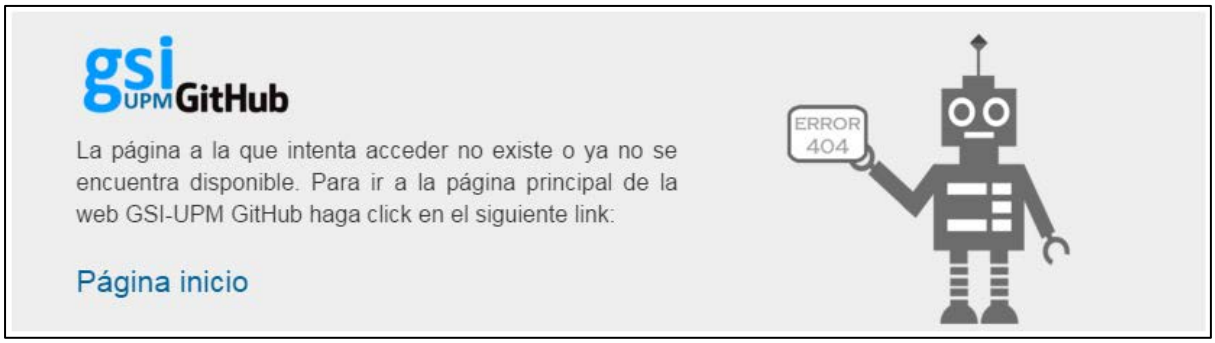

Figura 4.19: Página no encontrada

4.11. NOTIFICACIONES, EFECTO DE CARGA E ICONO

Algunas acciones realizadas en la web provocan la aparición de notificaciones en la parte inferior derecha de la pantalla una vez finalizadas. Guardar repositorios, actualizarlos, eliminar los que ya no existan y cerrar sesión son estas acciones. Se trata de un efecto logrado con CSS gracias al cual un contenedor aparece cinco segundos y transcurridos desaparece. Su aspecto se puede ver en la siguiente figura.

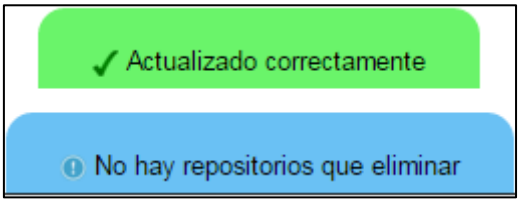

Figura 4.20: Notificaciones

Además se ha incluido un efecto de carga para que bloquee la pantalla y avise de que la página está realizando alguna acción. Este efecto puede verse al ejecutar acciones de administrador o al entrar en las páginas que realizan peticiones a Firebase.

Figura 4.21: Efecto de carga de la web

Por último, se ha diseñado un icono para la web que puede verse en la siguiente figura.

Figura 4.22: Icono de la web

4.12. VERSIÓN MÓVIL

Todo lo expuesto anteriormente se corresponde con la versión de escritorio. Si se utiliza un dispositivo móvil, la web se adapta al tamaño del ancho de la pantalla. A continuación se muestran capturas de la versión para pantallas reducidas.

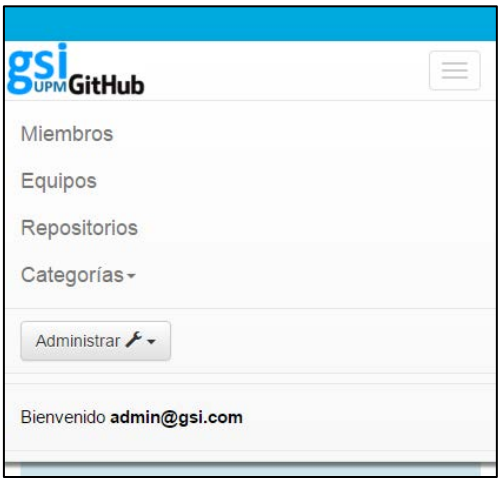

Figura 4.23: Versión móvil - Menú global

Figura 4.24: Versión móvil - Barra de administrador

Figura 4.25: Versión móvil - Página principal

4.13. RESUMEN

El resultado final de la web ha quedado definido y expuesto en este capítulo, al igual que las posibilidades que ofrece. Como se ha podido observar, el diseño está adaptado para su correcta visualización en versiones de escritorio y móviles gracias a Bootstrap. Este portal se ha ideado para ser actualizado de manera manual por el administrador cada pocos días con el objetivo de que la información que muestra no quede obsoleta. No obstante, en el siguiente capítulo se abordarán mejoras propuestas para evitar este hecho.

5. CONCLUSIONES Y LÍNEAS FUTURAS

5.1. CONCLUSIONES

El objetivo principal de este proyecto era el desarrollo de un portal web para los repositorios de GitHub pertenecientes al GSI. Desde el planteamiento inicial hasta el resultado obtenido se han realizado modificaciones en función de las limitaciones de las tecnologías utilizadas. Al mismo tiempo se han añadido funcionalidades que en un principio no se plantearon, como el almacenamiento online o la figura de un administrador de la web.

Jekyll ha resultado ser una herramienta cómoda y de fácil uso una vez entendido su funcionamiento. Si bien su utilización está más orientada a blogs o a páginas de información estática pura, ha demostrado cierta versatilidad para esta aplicación. La ausencia de *back-end* es quizás el mayor inconveniente pero, como se ha expuesto en este proyecto, se puede suplir usando una de las múltiples bases de datos online disponibles en el mercado.

El alojamiento en las Páginas de GitHub utilizado es una opción idónea para páginas web estáticas. Manejar el proyecto como un repositorio más, sin necesidad de conexiones FTP, es una de las principales ventajas que se ha encontrado. Como inconveniente, sólo es posible tener una web desplegada para cada cuenta de GitHub, por lo que su uso no es sería el más adecuado para un diseñador web.

La base de datos online Firebase ofrece muchas posibilidades hasta hace poco no exploradas. Poder almacenar los contenidos de una web en la 'nube' es una práctica que para ciertas aplicaciones como esta es una estrategia clave. Una de las características que le han convertido en una tecnología puntera es la detección automática de cambios en la base de datos. Esto quiere decir que si un usuario está visitando la web y se modifica algún contenido en Firebase, desde el lado del cliente se detecta y se actualiza la página. Es una funcionalidad dirigida a un posible chat o sistema de comentarios. En la web diseñada no se ha encontrado un objetivo en el que poder aplicar esta característica.

En los objetivos planteados en la introducción (Sec.1.2) se señaló que como objetivo de fondo se buscaba realizar una web preparada para posibles modificaciones futuras. Este planteamiento se ha conseguido en varios aspectos. Por un lado, como se ha indicado en la sección 3.4.2 de 'Páginas de repositorios por categorías', es posible añadir, editar o eliminar categorías con pocas modificaciones. Por otro lado, las variables determinantes como la cuenta de la organización de GitHub, el nombre de la base de datos de GitHub o las claves de acceso para la API de GitHub sólo se definen una vez en todo el proyecto. Con esto se pretendía facilitar el trabajo de cambiar algunos de esos valores en el futuro si fuera necesario.

El resultado: un portal web alojado en GitHub donde se puede consultar información administrada sobre repositorios, miembros y equipos de la organización de GitHub del GSI. Se puede visitar siguiendo el siguiente enlace:

http://gsi-upm.github.io/

5.2. LÍNEAS FUTURAS

Una vez finalizado el proyecto, se han analizado posibles mejoras y funciones que podrían ser de utilidad. Algunas se detectaron durante su desarrollo pero no han sido incluidas. A continuación se detallan las que se han considerado como principales.

- Añadir nuevos campos para filtrar repositorios, como el lenguaje principal, el tamaño base del proyecto o el perfil. Actualmente los filtros existentes en la web son en función de nombre, id, autor y fechas de creación y actualización.

- Incluir más campos de información para cada repositorio, como los lenguajes utilizados además del principal o un listado con los últimos *commits* realizados.

- Diseñar un sistema de automatización de actualización de la página. La página está pensada para ser actualizada periódicamente por el administrador de la web. Sin embargo, se ha pensado elaborar una lógica que permita desde la página de administración programar cada cuánto se debe actualizar la página. En Firebase se podría guardar esta configuración y cada vez que se abra la página comprobar si se debe actualizar el contenido o no.

- Mejorar visualmente algunas páginas como la de Equipos o Miembros e implementar búsquedas para los mismos.

- Arreglar algunas incompatibilidades con los elementos HTML de la página, que no afectan al funcionamiento de la web, pero puede provocar efectos indeseados.

6. BIBLIOGRAFÍA

[1] «Página oficial de Jekyll,» [En línea]. Available: http://jekyllrb.com/docs/home/.

[2] «Repositorio de Jekyll en GitHub,» [En línea]. Available: https://github.com/jekyll/jekyll.

[3] T. Preston-Werner, «Blogging Like a Hacker,» 17 Noviembre 2008. [En línea]. Available: http://tom.preston-werner.com/2008/11/17/blogging-like-a-hacker.html.

[4] «Página oficial de Jekyll-Bootstrap,» [En línea]. Available: http://jekyllbootstrap.com/.

[5] «Página GitHub de Jekyll-Bootstrap,» [En línea]. Available: https://github.com/plusjade/jekyll-bootstrap.

[6] «Introducción a Jekyll-Bootstrap,» [En línea]. Available: http://jekyllbootstrap.com/lessons/jekyll-introduction.html.

[7] «Página oficial de GitHub Desktop,» [En línea]. Available: https://desktop.github.com/.

[8] «Página oficial de GitHub Pages,» [En línea]. Available: https://pages.github.com/.

[9] «Página de la API de GitHub,» [En línea]. Available: https://developer.github.com/v3/repos/.

[10] «Página oficial de Firebase,» [En línea]. Available: https://www.firebase.com/docs/web/guide/.

[11] «Documentación de Liquid,» [En línea]. Available: https://shopify.github.io/liquid/basics/introduction/.

[12] «Página oficial de YAML,» [En línea]. Available: http://yaml.org/.

[13] «Página oficial de JSON,» [En línea]. Available: http://www.json.org/.

[14] «Página GitHub de Rundown,» [En línea]. Available: https://github.com/rubyworks/rundown.

[15] «Página GitHub de Showdown,» [En línea]. Available: https://github.com/showdownjs/showdown.

[16] «Página del elemento Toggle de Bootstrap,» [En línea]. Available: http://www.bootstraptoggle.com/.

[17] «Página de la librería DateTimePicker de Bootstrap,» [En línea]. Available: https://eonasdan.github.io/bootstrap-datetimepicker/.

[18] «Artículo de ayuda sobre GitHub, Liquid y Jekyll,» [En línea]. Available: https://help.github.com/articles/repository-metadata-on-github-pages/.

[19] «Building a Jekyll Site – Part 3 of 3: Creating a Firebase-Backed Commenting System,» 11 Febrero 2016. [En línea]. Available: https://css-tricks.com/building-a-jekyll-site-part-3-of-3/.

[20] «Documentación oficial de Firebase - Función val(),» [En línea]. Available: https://www.firebase.com/docs/web/api/datasnapshot/val.html.

[21] «Página oficial del GSI de la UPM,» [En línea]. Available: http://www.gsi.dit.upm.es/es/.

[22] «GitHub Desktop para Windows,» [En línea]. Available: https://desktop.github.com/.

[23] A. Debenham, «Get Started With GitHub Pages (Plus Bonus Jekyll),» 18 Diciembre 2013. [En línea]. Available: https://24ways.org/2013/get-started-with-github-pages/.

[24] «Página GitHub de RDoc,» [En línea]. Available: https://github.com/rdoc/rdoc.

[25] «Documentación oficial de Firebase,» [En línea]. Available: https://www.firebase.com/docs/web/guide/.