Universidad Zaragoza

Trabajo Fin de Grado

Baboon:
Gestor de campañas de email marketing

Autor

Carlos Martínez Castillero

Director

Raúl Novoa Mínguez

Escuela de Ingeniería y Arquitectura

2016

DEDICATORIA

A mis padres
Mª Rosa y Aniceto,
por todo vuestro incansable esfuerzo y dedicación.

A mi hermana
María,
por aportarme tanta felicidad.

A mi amigo Adam,
por tu apoyo incondicional.

A mi director y a mi ponente
Raúl Novoa y F.J. López Pellicer,
por darme la oportunidad de aprender tanto de vosotros.

A mi abuelo
Bernardo,
sé que estarías orgulloso de tu nieto.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

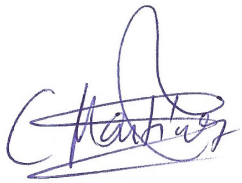
D./Dª. Carlos Martínez Castillero,

con nº de DNI 76973738-Y en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Baboon: gestor de campañas de email marketing,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 21 - Junio - 2016



Fdo: Carlos Martínez Castillero

Índice

Índice de figuras

Resumen ejecutivo

En el presente documento se quiere mostrar todo el trabajo realizado durante los últimos meses para conseguir convertir una serie de ideas propuestas en un producto software que va a dar servicio a un elevado número de usuarios.

A día de hoy, la existencia de Internet y las tecnologías actuales han hecho posible que la comunicación entre las personas sea tarea sencilla y que, además, esté al alcance de cualquiera.

Esta facilidad e inmediatez de llegar a la gente es la que ha despertado gran interés en las empresas estos últimos años, que ven en estas tecnologías una gran oportunidad de hacer llegar su producto o servicio al público, clientes o potenciales usuarios.

Es por este motivo que nace *Baboon*. En **10Labs** – empresa en la cual se realizó este trabajo – llevan ya algunos años desarrollando una solución concreta orientada al pequeño comercio y franquicias, dentro de la cual se habían planteado incorporar un servicio que permitiera el envío de ofertas y todo tipo de promociones a través del correo electrónico.

La memoria recoge todo el trabajo realizado y las tecnologías empleadas, empezando por las tareas de análisis llevadas a cabo, el posterior diseño y su implementación, hasta el despliegue del proyecto. Todo ello basándose en la metodología ágil de desarrollo de software y proyectos Scrum.

Para su elaboración se han utilizado tecnologías web modernas como AngularJS en la parte del cliente, Node.js y Express.js para dar servicio a los usuarios a través de una API Web y MongoDB para la persistencia de los datos.

El resultado es un servicio en la nube que permite el lanzamiento de campañas de email marketing, la visualización del impacto que ha tenido cada campaña lanzada y la gestión del contenido que se envía en cada correo mediante plantillas HTML de correo electrónico.

1 Introducción

1.1 Contexto

10Labs Digital Technology es una pequeña empresa que se dedica principalmente al desarrollo de soluciones software mediante la aplicación de las tecnologías web más actuales, caracterizadas por su agilidad, rapidez y flexibilidad.

Fundamentalmente desarrollan sus propios productos, como parte del equipo emprendedor de los proyectos que les plantean o en el desarrollo de soluciones a medida para clientes.

Uno de los productos propios de **10Labs** es *Fidelizoo*: un servicio de fidelización para pequeño comercio y franquicias. Entre las funcionalidades propias de este servicio se encuentra el envío de correos electrónicos personalizados a clientes o grupos de clientes que cada establecimiento que utiliza esta aplicación ha registrado en el sistema.

Esta funcionalidad sería muy útil poder extenderla a otros productos o que formase parte de futuras aplicaciones; tener al alcance de unos cuantos *clicks* de ratón la posibilidad de enviar correos electrónicos a un determinado grupo de personas y, además de esto, ser capaz de visualizar el impacto que ha tenido dicho envío es algo que podría resultar realmente interesante. El inconveniente hasta el momento es que este servicio, como se ha mencionado anteriormente, forma parte exclusivamente de un producto concreto de forma limitada y, por lo tanto, a otras soluciones que quisiesen hacer uso de él les sería imposible de esta manera.

Es por ello que la forma de resolver este problema pasaría por desarrollar un servicio independiente cuya única responsabilidad fuese la gestión íntegra de campañas de email marketing. De esta forma dispondríamos de una única solución de la cual podrían hacer uso tantas aplicaciones externas como necesitasen integrar esta funcionalidad, en vez de tener que desarrollarla tantas veces como productos la requiriesen.

Dicho así, hablamos del modelo de distribución de software conocido como *Software as a Service.* Para el desarrollo de la solución será necesario el empleo de tecnologías web que nos permitan la integración del servicio cuando sea necesario a través de un API Web.

1.2 Objetivo del proyecto

El proyecto planteado cumple sobradamente los requisitos de un proyecto de Ingeniería lo suficientemente complejo como para que un alumno pueda poner en práctica todos los conocimientos que ha adquirido a lo largo de toda su etapa como estudiante de Ingeniería Informática y, especialmente, aquellos adquiridos durante sus últimos años de especialidad en Ingeniería de Software debido a la naturaleza del problema.

El objetivo principal y más importante que se desea conseguir desarrollando este Trabajo Fin de Grado es, por encima de todo, aprender lo máximo posible. Las características del trabajo a realizar va a permitir el aprendizaje y utilización de algunas de las tecnologías más actuales, interesantes y demandadas en el mundo profesional, y el desarrollo de una solución real que va a ser utilizada para ofrecer un servicio a clientes reales.

Además de adquirir conocimientos técnicos importantes, todo el proceso de desarrollo del proyecto hará posible que el alumno adquiera también cierta experiencia y soltura en la gestión de proyectos software: planificación, estimación, análisis… cualidades muy significativas, propias de un ingeniero de software.

1.3 Marco tecnológico y profesional

El correo electrónico (en inglés: *e-mail*), es un servicio de red que permite a los usuarios enviar y recibir mensajes; todo esto a través de Internet. Es por ello, que cuando hablemos de las tecnologías que rodean a este fenómeno, estaremos hablando de tecnologías que permitan hacer uso de la red para realizar el envío de estos mensajes.

Hoy en día, existen todo tipo de servicios para la gestión masiva de correos, y realizar envíos de correos electrónicos transaccionales. Todas estas herramientas, generalmente, tienen algo en común: ofrecen el servicio a través de sus propias aplicaciones web o aplicaciones nativas (Android, iOS…) y, además, permiten su integración a través de su propio API Web de forma segura.

Por esta razón, las tecnologías asociadas a este tipo de sistemas son las denominadas tecnologías web: aquellas que hacen uso de todas las tecnologías para la interconexión de computadoras y de presentación – interfaces gráficas de usuario que permiten presentar su funcionamiento de forma sencilla e intuitiva a éste.

1.4 Estructura de la memoria

Este documento se ha estructurado en cuatro secciones principales, enumeradas a continuación: **Introducción** (sección en la que se encuentra esta subsección), **Trabajo realizado**, **Gestión del proyecto** y **Conclusiones**. Aparte de estos cuatro bloques donde se recoge todo lo fundamental del trabajo realizado, también existen otros dos más: **Bibliografía** y **Anexos**.

La sección de **Introducción** nos ofrece una breve descripción del problema a abordar y sitúa al lector en contexto de forma que, tras realizar una primera lectura, habrá adquirido una visión del proyecto planteado, además de conocer los objetivos que pretenden conseguirse con este trabajo y el marco tecnológico en el que se sitúa.

A continuación, se sitúa la sección **Trabajo realizado**. Aquí se encuentra recogido todo el trabajo técnico y de Ingeniería llevado a cabo a lo largo de todo el proceso de desarrollo del proyecto: desde el análisis del problema hasta el despliegue del producto, pasando por el diseño de cada uno de los componentes software, la arquitectura, implementación, verificación y validación del sistema. También se detallan las tecnologías empleadas y para qué han sido utilizadas exactamente.

Posteriormente, en tercer lugar, encontraremos todo aquello relacionado con la metodología empleada en la gestión del Trabajo Fin de Grado, en la sección **Gestión del proyecto**. Asimismo, este bloque congrega las herramientas que han formado parte de este proceso concreto y todos los detalles de su planificación.

El último de los cuatro principales bloques, lo constituye la sección de **Conclusiones**. Esta parte recoge los resultados obtenidos tras la finalización del trabajo, las lecciones aprendidas y una conclusión personal del alumno.

A la sección anterior le sigue la **Bibliografía**. Aquí se enumeran las fuentes que han sido consultadas durante el desarrollo del proyecto y de este documento.

En ultimo lugar, encontraremos los **Anexos**. Este bloque contiene documentación adicional relacionada con la previamente expuesta, y todos ellos son una extensión más detallada de algunas de las secciones anteriores. A continuación se expondrán junto con una breve descripción los anexos que amplían este documento:

- **Anexo I: Casos de Uso**
  - Recoge una descripción en profundidad y detallada de los Casos de Uso del sistema.
- **Anexo II: Requisitos funcionales**
  - Recoge una especificación en profundidad y detallada de los Requisitos funcionales del sistema.
- **Anexo III: Requisitos no funcionales**
  - Recoge una especificación en profundidad y detallada de los Requisitos no funcionales del sistema.
- **Anexo IV: BPMN Crear plantilla**
  - Se detalla el flujo que sigue la creación de una plantilla de correo electrónico mediante un diagrama BPMN.
- **Anexo V: BPMN Envío correo electrónico**
  - Se detalla el flujo que sigue el envío de un correo electrónico perteneciente a una plantilla mediante un diagrama BPMN.
- **Anexo VI: Diagrama de estados campañas**
  - Se detallan los distintos estados por los que puede pasar una campaña mediante un diagrama de estados.
- **Anexo VII: Diagrama de estados correos electrónicos**
  - Se detallan los distintos estados por los que puede pasar un correo electrónico perteneciente a una campaña.
- **Anexo VIII: Documentación del API Web**
  - Descripción y documentación en gran nivel de detalle, de las APIs Web implementadas en el sistema.
- **Anexo IX: EditorConfig**
  - Descripción de la herramienta *EditorConfig* utilizada en el proceso de desarrollo del trabajo.
- **Anexo X: JSON Web Tokens**
  - Explicación del funcionamiento de los JSON Web Token y las partes que los componen.
- **Anexo XI: Tests**
  - Recoge la especificación de todos los tests implementados en el sistema.
- **Anexo XII: Configuración por entornos**
  - Se recogen todos los detalles relacionados con la configuración realizada por entornos en el proyecto.
- **Anexo XIII: Planificación inicial SPRINT-1**

    - Documento que recoge todos los detalles de la planificación inicial realizada para llevar a cabo en el SPRINT-1.
- **Anexo XIV: Replanificación SPRINT-1**
    - Documento que recoge todos los detalles de la replanificación realizada para llevar a cabo en el SPRINT-1.
- **Anexo XV: Planificación SPRINT-2**
    - Documento que recoge todos los detalles de la planificación realizada para llevar a cabo en el SPRINT-2.
- **Anexo XVI: Planificación SPRINT-3**
    - Documento que recoge todos los detalles de la planificación realizada para llevar a cabo en el SPRINT-3.
- **Anexo XVII: Planificación SPRINT-4**
    - Documento que recoge todos los detalles de la planificación realizada para llevar a cabo en el SPRINT-4.

2 Trabajo realizado

2.1 Análisis

2.1.1 Contexto

En esta sección se recoge todo el trabajo de análisis realizado. En primer lugar se describen los Casos de Uso. A continuación, se describen la funcionalidades en forma de requisitos: funcionales y no funcionales. Para finalizar, se detallan las tecnologías concretas utilizadas durante el proyecto y herramientas de desarrollo empleadas.

2.1.2 Casos de Uso

A continuación se muestran y describen cada uno de los casos de uso para cada tipo de usuario que puede interactuar con el sistema. Principalmente, se distinguen dos tipos de usuarios:

- **Administrador:** usuario encargado de realizar tareas de administración en el sistema.
- **Usuario aplicación:** usuario que hace uso de las principales y fundamentales funcionalidades del sistema.

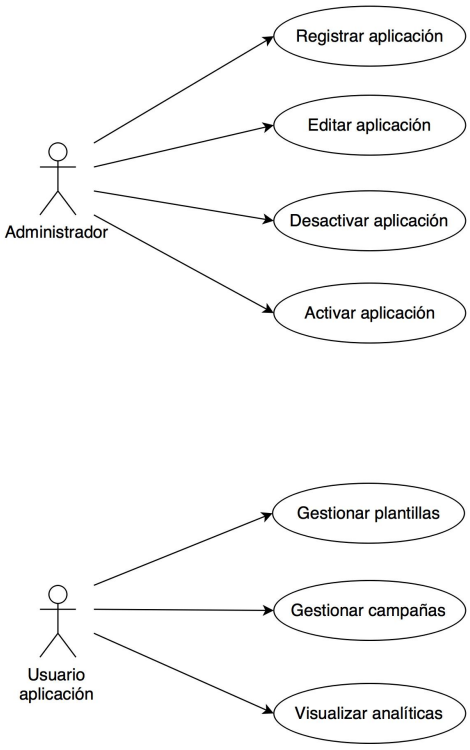


Figura 1. Casos de uso básicos

En el diagrama anterior, se presentan los casos de uso básicos. Puede encontrarse una visión más profunda y detallada de esta sección en el **Anexo I: Casos de Uso**.

Registrar aplicación

Un usuario administrador se identifica, accede al sistema y solicita dar de alta a una aplicación. El sistema le confirma el alta de la aplicación tras comprobar que la

operación puede realizarse, y a partir de este momento la aplicación registrada podrá hacer uso de los servicios que ofrece la plataforma.

Editar aplicación

Un usuario administrador se identifica, accede al sistema y solicita modificar la información asociada a una de las aplicaciones dadas de alta. El sistema solicita al administrador los cambios que quiere hacer, éste los introduce. El sistema confirma la modificación tras comprobar que la operación puede realizarse y termina el caso de uso.

Desactivar aplicación

Un usuario administrador se identifica, accede al sistema y solicita desactivarle el acceso al servicio que ofrece la plataforma a una de las aplicaciones dadas de alta. El sistema confirma la operación tras comprobar que puede realizarse y, a partir de este momento, la aplicación no podrá hacer uso de los servicios que ofrece la plataforma.

Activar aplicación

Un usuario administrador se identifica, accede al sistema y solicita activarle el acceso al servicio que ofrece la plataforma a una de las aplicaciones dadas de alta. El sistema confirma la operación tras comprobar que puede realizarse y, a partir de este momento, la aplicación podrá volver a hacer uso de los servicios que ofrece la plataforma.

Gestionar plantillas

Un usuario aplicación se identifica y solicita gestionar sus plantillas de correo electrónico. El sistema solicita al usuario que especifique qué operación de las disponibles desea realizar. El usuario selecciona la operación que desea realizar y acompaña la selección con los datos que la requieran hacer efectiva. El sistema confirma la operación tras comprobar que puede realizarse y termina el caso de uso.

Gestionar campañas

Un usuario aplicación se identifica y solicita gestionar sus campañas de email marketing. El sistema solicita al usuario que especifique qué operación de las disponibles desea realizar. El usuario selecciona la operación que desea realizar y acompaña la selección con los datos que la requieran hacer efectiva. El sistema confirma la operación tras comprobar que puede realizarse y termina el caso de uso.

Visualizar analíticas

Un usuario aplicación se identifica y solicita visualizar las analíticas de sus campañas de email marketing. El sistema solicita al usuario qué campaña de las disponibles desea seleccionar y éste selecciona la campaña deseada. El sistema confirma la operación tras comprobar que puede realizarse y le muestra al usuario las analíticas de la campaña solicitada.

2.1.3 Requisitos funcionales

Código	Requisito
RF-1	El usuario administrador deberá ser capaz de dar de alta aplicaciones en el sistema.
RF-2	El usuario administrador deberá ser capaz de dar de modificar la información de cualquier aplicación registrada en el sistema.
RF-3	El usuario administrador deberá ser capaz de dar de visualizar un listado de las aplicaciones registradas en el sistema.
RF-4	El usuario administrador deberá ser capaz de dar de denegar la utilización del servicio a cualquier aplicación registrada en el sistema.
RF-5	El usuario administrador deberá ser capaz de dar de habilitar la utilización del servicio a cualquier aplicación registrada en el sistema a la que se lo haya denegado previamente.
RF-6	El usuario aplicación deberá ser capaz de registrar plantillas de correo-electrónico.
RF-7	El usuario aplicación deberá ser capaz de modificar la información de cualquiera de sus plantillas.
RF-8	El usuario aplicación deberá ser capaz de eliminar cualquiera de sus plantillas.
RF-9	El usuario aplicación deberá ser capaz de dar de visualizar un listado de todas sus plantillas.
RF-10	El usuario aplicación deberá ser capaz de crear campañas de email marketing.
RF-11	El usuario aplicación deberá ser capaz de crear campañas de email marketing a partir de otras ya existentes, obteniendo una nueva campaña idéntica a otra que haya sido lanzada previamente.
RF-12	El usuario aplicación deberá ser capaz de modificar la información de cualquiera de sus campañas que no hayan sido todavía lanzadas.
RF-13	El usuario aplicación deberá ser capaz de eliminar campañas no lanzadas y archivar campañas ya enviadas.
RF-14	El usuario aplicación deberá ser capaz de enviar las campañas que haya creado pero no enviado previamente.
RF-15	El usuario aplicación deberá ser capaz de visualizar un listado de todas sus campañas.

RF-16	El usuario aplicación deberá ser capaz de visualizar el impacto de las campañas: total de correos enviados, pendientes de envío, rechazados, marcados como spam y abiertos.

En el **Anexo II: Requisitos funcionales** se encuentra organizado por tipo de usuario, el catálogo de requisitos funcionales de forma detallada.

2.1.4 Requisitos no funcionales

Código	Requisito
RNF-1	Los usuarios administradores deberán hacer uso del sistema a través de un cliente web.
RNF-2	Los usuarios aplicación deberán hacer uso de los servicios del sistema a través de una API Web.
RNF-3	La API Web que utilizarán los usuarios aplicación deberá estar securizada mediante un par de claves: API Key y API Secret.

En el **Anexo III: Requisitos no funcionales** se encuentra el catálogo de requisitos no funcionales de forma detallada.

2.1.6 Modelo de dominio

Partiendo del catálogo de requisitos, se construyó un primer modelo del dominio mediante un diagrama de clases.

A continuación, se muestra el diagrama de clases básico que ha sido obtenido a partir del análisis previo realizado:

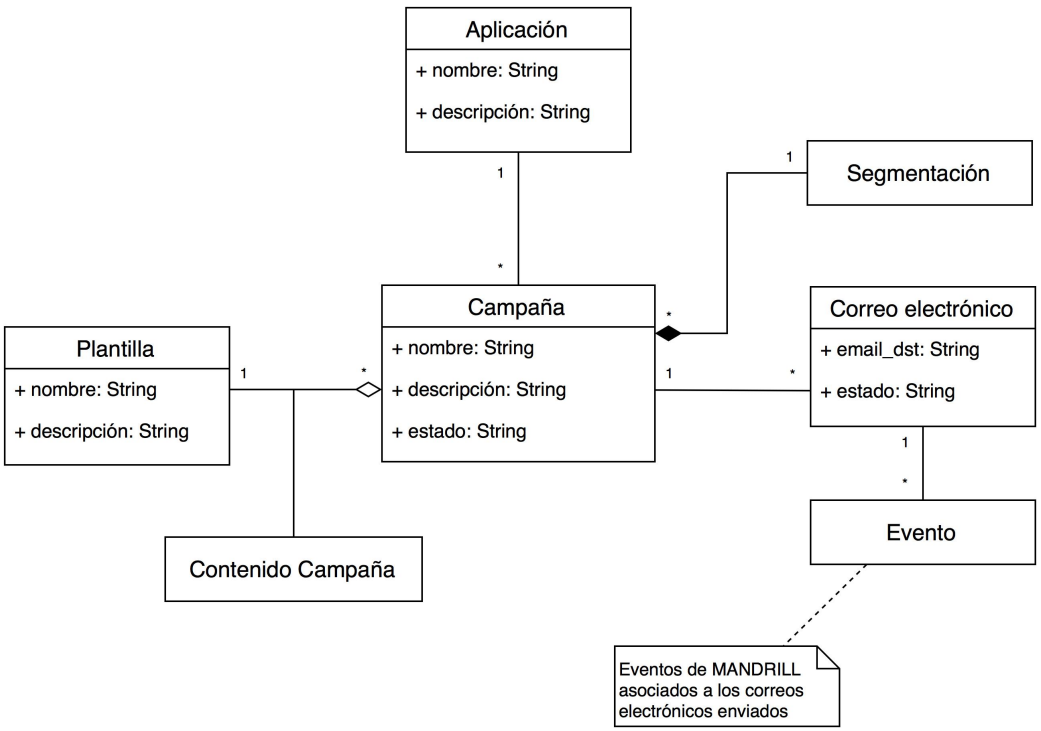


Figura 2. Modelo de dominio

Este diagrama nos será útil, además, a la hora de diseñar el modelo de datos que va a tener nuestra aplicación.

2.1.7 Tecnologías

Se ha hecho uso de tecnologías web modernas. Específicamente se ha utilizado el **MEAN Stack**: una colección de tecnologías Javascript (**M**ongoDB, **E**xpress.js, **A**ngularJS y **N**ode.js) empleadas para el desarrollo de aplicaciones web. Desde la parte del cliente y la del servidor hasta la persistencia de la aplicación, estamos ante una herramienta de desarrollo *full-stack*, donde desarrollamos una solución de software completa utilizando un mismo lenguaje de programación.

El **MEAN Stack** lo constituyen, concretamente:

- **MongoDB[1]:** Sistema de base de datos orientado a documentos, que guarda estructuras de datos en documentos similares a JSON con un esquema dinámico. Utiliza una especificación llamada BSON, haciendo que la integración de los datos en ciertas aplicaciones sea más rápida y sencilla.
- **Express.js[2]:** Framework minimalista y flexible para el desarrollo de aplicaciones web en *Node.js* que dispone de una gran variedad de métodos y utilidades sobre el protocolo *http*, así como distintos tipos de middleware que permite la creación de un API Web robusto fácilmente y con rapidez.
- **AngularJS[3]:** Framework mantenido principalmente por Google que se utiliza principalmente para el desarrollo de aplicaciones web de una sola página (*Single Page Applications*) con capacidad de Modelo-Vista-Controlador (MVC). Adapta y amplía el HTML tradicional para servir contenido dinámico a través de un *data binding* bidireccional que permite la sincronización automática de modelos y vistas.
- **Node.js[4]:** Entorno en tiempo de ejecución multiplataforma para la capa del servidor, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Fue creado con enfoque a ser útil en la creación de programas de red altamente escalables como, por ejemplo, servidores web.

Para implementar los tests del proyecto se han utilizado una serie de frameworks y librerías Javascript que nos permiten testear código que se ejecuta de manera asíncrona, incluir aserciones en los tests e incluso testear nuestro API Web incluyendo, también, aserciones a nivel de protocolo *http*. En esta parte del proyecto se ha utilizado, en particular:

- **Mocha[5]:** Framework de testing que puede ser ejecutado tanto en el navegador como en un entorno de *Node.js*. Nos permite testear código asíncrono y describir de forma sencilla colecciones de tests, ejecutándolas de manera secuencial ofreciéndonos un sistema de reporte flexible y preciso.

---

[1] https://www.mongodb.com

[2] http://expressjs.com

[3] https://angularjs.org

[4] https://nodejs.org

[5] https://mochajs.org

- **Should.js[6]:** Librería de aserciones expresiva, legible e independiente del framework de testing donde se emplee.
- **Supertest[7]:** Libería de aserciones a nivel de protocolo *http* cuyo principal objetivo es proveer un alto nivel de abstracción en el testing de APIs Web.

Además de lo expuesto anteriormente, cabe enumerar varias librerías o módulos importantes de Javascript que se han empleado también en el proyecto. Estas son:

- **Q[8]:** Librería para la creación y composición de promesas asíncronas en Javascript.
- **Q-io[9]:** Interfaz de IO (sistema de ficheros, *http*) que hace uso de promesas. Hace uso de la librería anteriormente nombrada, *Q*.
- **Winston[10]:** *Logger* multi transporte y asíncrono para *Node.js*. Permite crear varias instancias y configurar cada una de ellas de manera independiente.
- **Lodash[11]:** Librería moderna de utilidades Javascript que ofrece funciones para trabajar, iterar y manipular listas, números, objetos, cadenas, etc. entre otras.
- **Express-JWT[12]:** Middleware para *Express.js* que permite la autenticación de peticiones *http* utilizando *JWT tokens* en aplicaciones *Node.js*.
- **ApidocJS[13]:** Librería que permite la generación de documentación de APIs Web RESTful a partir de anotaciones en el código.

2.1.8 Herramientas de desarrollo

Durante la fase implementación se ha hecho uso, además, de varias herramientas que facilitan al desarrollador agilizar el proceso permitiéndole, entre otras, gestionar dependencias de manera inmediata o llevar a cabo la automatización y ejecución de tareas. Se han utilizado en este caso:

- **Npm[14]:** Gestor de paquetes de Javascript que se utiliza para instalar, compartir y distribuir código; permite gestionar de forma sencilla e inmediata las dependencias en proyectos. El registro de *npm* contiene alrededor de 250.000 módulos de código reutilizable, convirtiéndolo en uno de los registros más grandes del mundo.
- **Bower[15]:** Gestor de paquetes para la Web, optimizado especialmente para *frontend*. Permite gestionar las dependencias que van a utilizarse en un proyecto web como pueden ser frameworks, librerías, utilidades; principalmente componentes que contienen HTML, CSS, Javascript, fuentes o incluso ficheros de imágenes.

---

[6] https://shouldjs.github.io

[7] https://github.com/visionmedia/supertest

[8] http://documentup.com/kriskowal/q

[9] http://documentup.com/kriskowal/q-io

[10] https://github.com/winstonjs/winston

[11] https://lodash.com

[12] https://github.com/auth0/express-jwt

[13] http://apidocjs.com

[14] https://www.npmjs.com

[15] http://bower.io

- **Grunt[16]:** Herramienta que permite llevar a cabo la automatización de tareas repetitivas como minificación de ficheros CSS o Javascript, compilación, lanzamiento de tests, etc. para poder lanzarlas de manera instantánea siempre que sea necesario. *Grunt* dispone de cientos de *plugins* entre los que se encuentran muchas de las tareas que un desarrollador puede necesitar automatizar, los cuales permiten que configurar y realizar su lanzamiento resulte ser algo prácticamente inmediato y, además, que no requiera ningún tipo de esfuerzo.

El gestor de paquetes **npm** se ha empleado exclusivamente para gestionar las dependencias del *backend* del proyecto. A través de una serie de comandos por consola y un fichero de configuración (*package.json*) permite añadir y eliminar dependencias de forma realmente sencilla.

Por otro lado, **bower** se ha utilizado para gestionar las dependencias del *frontend*. La utilización de esta herramienta es prácticamente idéntica a la de **npm**: permite añadir y eliminar dependencias mediante comandos por consola y un fichero de configuración (*bower.json*).

Finalmente, **grunt** ha sido de gran utilidad y ayuda durante todo el proceso de desarrollo del proyecto permitiendo realizar el lanzamiento de una serie de tareas importantes tan solo con introducir sencillos comandos. Entre esta serie de tareas caben destacar las siguientes:

- La automatización y ejecución de los tests del sistema. Muy útil e imprescindible conforme se iban implementando nuevas funcionalidades.
- Integración de todas las dependencias del *frontend* en el fichero HTML principal del proyecto (*index.html*).
- Empaquetamiento del proyecto para su despliegue automático en entorno de producción.

[16] http://gruntjs.com

2.2 Diseño

2.2.1 Arquitectura

La arquitectura del sistema desarrollado está basada en el modelo de aplicación distribuida *cliente-servidor*, como puede verse a continuación en el siguiente diagrama de componentes-conectores:

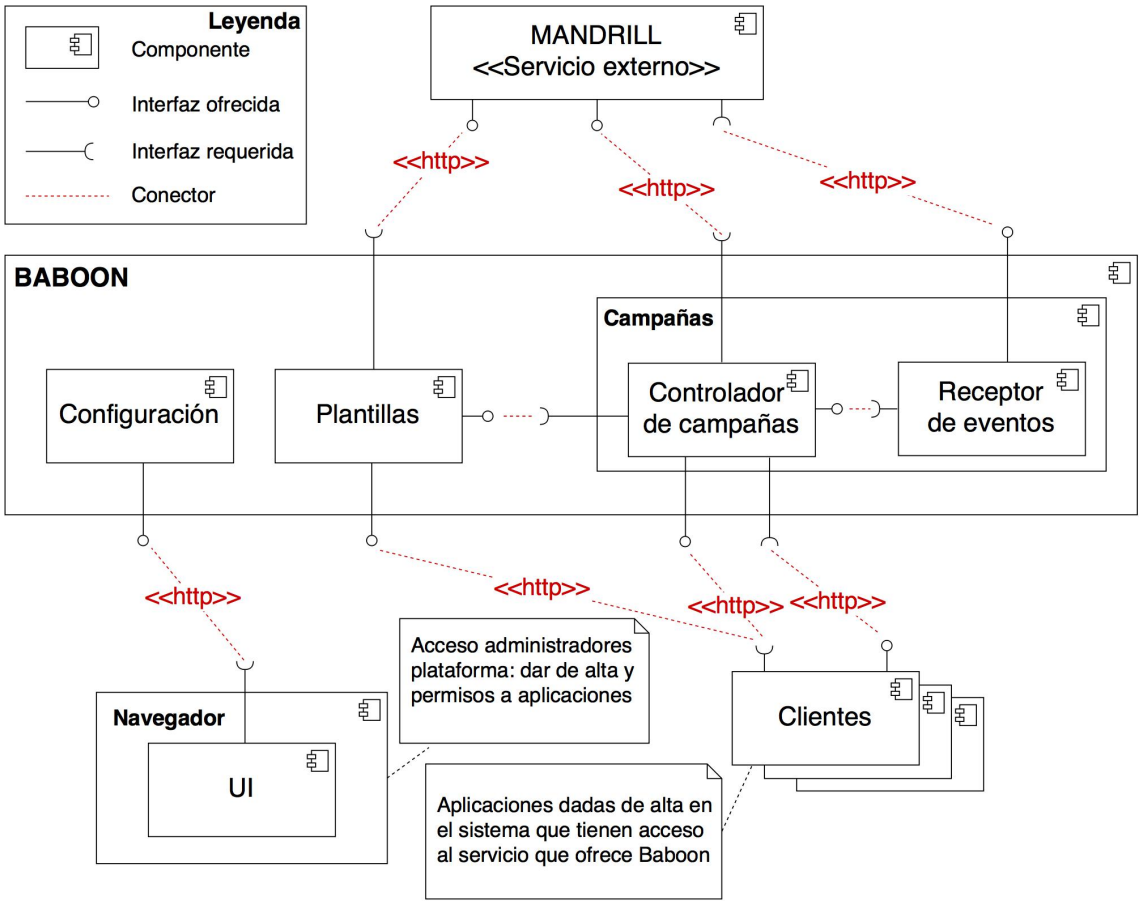


Figura 3. Arquitectura

Una buena parte del componente *BABOON* actúa como *servidor,* ofreciendo un conjunto de servicios a través de una API REST que son consumidos mayormente por las aplicaciones a las que previamente se les ha proporcionado permiso para hacerlo.

A su vez, *BABOON* actúa también como *cliente* del servicio externo – servicio en la nube escogido para realizar el envío seguro de correos electrónicos transaccionales, llamado ***Mandrill*** [17]– consumiendo su API REST.

Por otro lado, las tareas de administración llevadas a cabo por los usuarios correspondientes van a realizarse a través de un *cliente* web que consume un API REST interno. Esta API interna solamente será accesible desde el *frontend* implementado.

[17] https://www.mandrill.com

Las APIs REST, utilizarán en todo momento el formato *JSON* para el intercambio de datos y harán uso de los códigos de errores http correspondientes para dar respuesta a las peticiones.

2.2.2 Diseño del componente 'Configuración'

El componente *Configuración* se encarga de la gestión de la plataforma a nivel de administración. Sus responsabilidades son:

- Dar de alta aplicaciones en el sistema.
- Permitir/denegar servicio a aplicaciones dadas de alta.
- Listar las aplicaciones dadas de alta en el sistema y visualizar la información asociada a ellas.
- Modificar cualquier información de aquellas aplicaciones dadas de alta en el sistema.

Toda esta funcionalidad estará expuesta mediante una API REST interna, es decir, diseñada para ser utilizada en exclusiva por el *cliente web* desarrollado.

2.2.3 Diseño del componente 'Plantillas'

El componente *Plantillas* se encarga de la gestión de las plantillas que se utilizan para el envío de las campañas de correo. Sus responsabilidades son:

- Registrar plantillas en el sistema para su posterior utilización en los correos electrónicos enviados de cada campaña.
- Eliminar plantillas del sistema.
- Listar las plantillas registradas en el sistema y visualizar la información asociada a ellas.
- Modificar cualquier información de aquellas plantillas registradas en el sistema.

Toda esta funcionalidad estará expuesta mediante una API REST externa, es decir, diseñada para ser utilizada por las aplicaciones dadas de alta en el sistema que les esté permitido hacer uso del servicio.

2.2.4 Diseño del componente 'Campañas'

El componente *Campañas* se divide en dos subcomponentes que se detallan a continuación en las siguientes subsecciones: **2.2.4.1 Subcomponente 'Controlador campañas'** y **2.2.4.2 Subcomponente 'Receptor de eventos'**.

2.2.4.1 Subcomponente 'Controlador campañas'

El subcomponente *Controlador campañas* se encarga de la gestión y envío de las campañas de email marketing. Sus responsabilidades son:

- Creación de campañas de email marketing.
- Creación de campañas a partir de otras campañas existentes.
- Lanzamiento de campañas.
- Listar las campañas creadas y visualizar la información asociadas a éstas.
- Modificar cualquier información asociada a las campañas existentes.
- Eliminar o archivar campañas.
- Visualizar el impacto de campañas lanzadas.

Toda esta funcionalidad estará expuesta mediante una API REST externa, es decir, diseñada para ser utilizada por las aplicaciones dadas de alta en el sistema que les esté permitido hacer uso del servicio.

También requerirá que aquellas aplicaciones que hagan uso del servicio expongan una API REST para realizar la carga de destinatarios previa al lanzamiento de una campaña.

2.2.4.2 Subcomponente 'Receptor de eventos'

El subcomponente *Receptor de eventos* se encarga de la gestión de los eventos que cada correo electrónico enviado genera. Su responsabilidades son:

- Recepción de los eventos asociados a cada correo electrónicos que sean enviados desde el servicio externo.
- Determinar la campaña a la que pertenece cada evento recibido y delegar el registro del evento en el sistema al *Controlador de campañas*.

Deberá exponer un API REST de la cual el servicio externo hará uso para registrar en el sistema los eventos de los correos electrónicos.

2.2.5 API REST

En la siguiente tabla se recogen las diferentes rutas base que se han diseñado para las APIs REST, y el componente al que está asociada cada una, junto con una breve descripción.

Ruta base	Componente	Descripción
`/api/web/admin`	Configuración	Gestión de los usuarios administradores a los que se les permite dar de alta aplicaciones.
`/api/web/application`	Configuración	Gestión de las aplicaciones a las que se les permite hacer uso del servicio.
`/api/ext/v0.1/template`	Plantillas	Gestión de las plantillas.
`/api/ext/v0.1/campaign`	Controlador de campañas	Gestión de las campañas de email marketing.
`/api/webhook/event`	Receptor de eventos	Registro de los eventos asociados a los correos electrónicos enviados.

Se encontrará la documentación completa con todos los métodos de las APIs en el **Anexo VIII: Documentación del API Web**.

2.3 Implementación

2.3.1 Entorno de desarrollo

La primera mitad de la fase de implementación se llevó a cabo en una máquina virtual de *Ubuntu 14.02 LTS (64 bits)*, sobre un *Windows 7* y se hizo con **VirtualBox**, una aplicación multiplataforma de virtualización que extiende las capacidades de una computadora de tal forma que le permita correr multiples sistemas operativos (dentro de múltiples máquinas virtuales) al mismo tiempo. La segunda mitad, se realizó en una computadora sobre un *OS X*.

Para programar se ha utilizado **Sublime Text**[18], un editor de texto también multiplataforma, y el plugin **EditorConfig**[19] que ayuda a los desarrolladores a definir y mantener estilos de codificación consistentes entre diferentes editores e IDEs. El **Anexo IX: EditorConfig** contiene más detalles sobre su utilización en el proyecto.

2.3.2 Uso de guías de estilo

Se ha seguido, concretamente para la implementación del *frontend*, una guía de estilo de Angular JS. Particularmente la *Angular 1 Style Guide*[20] de John Papa[21], Desarrollador Experto de Google y *Regional Director* en Microsoft.

Generalmente, detrás de cada decisión y convención recogida en una guía de estilo suele existir un por qué. Hacer uso de una buena guía suele ser positivo a la larga debido a que son propensas a la aplicación de buenas prácticas, lo que se traduce en acabar teniendo una buena estructuración de la aplicación.

2.3.3 Utilización de promesas Javascript

Se ha sustituido el uso típico de *callbacks* por la utilización de promesas. Esto permite que los métodos asíncronos devuelvan valores como métodos síncronos: en lugar del valor final, el método asíncrono devuelve una *promesa* de tener un valor en algún momento del futuro.

Cuando se da la situación en la que hay que concatenar una serie de eventos asíncronos y ejecutarlos en orden secuencial uno detrás de otro, la utilización de *callbacks* acaba ocasionando el famoso '*Callback hell*', donde comienza a haber funciones anidadas dentro de otras, y el código entre otras cosas pierde legibilidad y dificulta su mantenibilidad, como se puede observar en el siguiente ejemplo:

[18] http://www.sublimetext.com

[19] http://editorconfig.org

[20] https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md

[21] https://johnpapa.net

```
obj.metodo1(data, function(data1) {
   obj.metodo2(data1, function(data2) {
      obj.metodo3(data2, function(data3) {
         obj.metodo4(data3, function(data4) {
            // etc
         });
      });
   });
});
```

Las promesas, por el contrario, nos permiten encadenar estas funciones mediante una serie de métodos que ofrecen, dejando nuestro código mucho más legible y mantenible, como puede apreciarse en la implementación del ejemplo anterior haciendo uso de promesas:

```
obj
 .metodo1(data)
 .then(obj.metodo2(data1))
 .then(obj.metodo3(data2)
 .then(obj.metodo4(data3))
 // etc.
```

2.3.4 Securización APIs ofrecidas

A continuación se expone cómo se ha implementado la seguridad de las APIs Web implementadas. En primer lugar se presenta la securización de la API interna de baboon, de la que hace uso exclusivo el cliente web. A continuación, se presenta la securización de la API externa que utilizan las aplicaciones dadas de alta. Por último, la de la API expuesta al servicio externo.

2.3.4.1 API interna

La API interna del sistema se ha securizado mediante *JSON Web Tokens (JWT)*.

El usuario se autentica en nuestra aplicación con un par de *usuario-contraseña*. Si esta pare es correcto, el servidor devuelve un token y a partir de entonces cada petición http que haga el usuario va acompañada con este token, que no es más que una firma cifrada que permite a nuestro API identificar al usuario.

El este token se almacena en el lado del cliente (en este caso, es la aplicación *AngularJS* la que lo almacena en el *localStorage* del navegador) y es el servidor el encargado de descifrar en cada petición ese token y redirigir el flujo de la aplicación en un sentido u otro.

Para validación del token se ha utilizado **ExpressJWT** (mencionado en la sección **2.1.7 Tecnologías**), un middleware desarrollado por **Auth0**, un equipo que desarrolla soluciones de autenticación.

Para una visión en más detallada de cómo funcionan los *JWT*, el **Anexo X: JSON Web Tokens** recoge en más profundidad este tipo de autenticación.

2.3.4.2 API externa

Para la securización de la API externa del sistema se ha implementado un middleware de autenticación propio. Cuando un usuario administrador da de alta una aplicación en el sistema se generan un par de claves asociadas a ésta, un `API Key` y un `API Secret`, que se utilizarán para llevar a cabo su autenticación.

En cada petición http realizada a esta API deben de existir las siguientes cabeceras:

Cabecera http	Descripción
`Baboon-timestamp`	Momento en el que se envía la petición http, en formato *UTC - Unix Timestamp.*
`Baboon-nonce`	Cadena de caracteres aleatoria.
`Baboon-apikey`	`API Key` de la aplicación que está utilizando la API.
`Baboon-authorization`	Firma de la petición; hash resultante en **Base64** de aplicar el algoritmo *SHA1* sobre una cadena, que llamaremos `keyString`, utilizando el `API Secret` como clave para su generación. El mecanismo de construcción del `keyString` a partir de la que se genera el hash, lo deberá de conocer tanto el servidor para validar las peticiones como las aplicaciones para firmarlas.

Construcción del keyString

La cadena a partir de la cual generaremos la firma de las peticiones, es la resultante de concatenar la ruta a la que se envía la petición, el *nonce* y el *timestamp* mediante el carácter almohadilla (*#*), y añadiendo a su vez a dicha concatenación dos caracteres almohadilla (*##*) más al final.

De esto se encarga la siguiente función:

```
// Genera la cadena a partir de la cual se genera la firma de una petición.
//
// Pre:<data.pathname>, <data.nonce> y <data.timestamp> deben de tener valor.
function createKeyString(data) {
  return data.pathname.concat('#')
    .concat(data.nonce)
    .concat('#')
    .concat(data.timestamp)
    .concat('##');
};
```

Posible ejemplo de `keyString` tras ejecutar la función:

`http://localhost:9000/api/ext/v0.1/template#vjbyPxybdZaNmG#1466613201642##`

Validación de las peticiones

En primer lugar, se comprueba que en la petición están definidas las cuatro cabeceras requeridas, que todos ellos tienen valor y que el valor de `Baboon-timestamp` se encuentra entre las últimas 24 horas.

Si esto se cumple, lo siguiente que se hace es obtener el `API Secret` de la aplicación cuyo `API Key` coincida con el especificado en `Baboon-apikey`. Se genera el `keyString` a partir de las cabeceras, se obtiene el hash resultante de aplicar el algoritmo correspondiente sobre dicha cadena utilizando el `API Secret` obtenido como clave, se codifica en **Base64** y se comprueba que su valor coincide con la firma de la petición que se encuentra en la cabecera `Baboon-authorization`.

Si la firma generada en el servidor coincide con la enviada por el cliente, se comprueba finalmente que el *nonce* especificado no ha sido registrado previamente en el sistema. En caso afirmativo, el middleware añade el objeto aplicación obtenido a partir del `API Key` especificado en la petición http, y permite que el flujo de ejecución continúe.

En caso contrario, se responde con el código de error correspondiente.

Observaciones

- El **`API Secret`** únicamente debe de conocerlo la entidad servidor y la cliente correspondiente, ya que se utiliza para poder firmar las peticiones. Es la principal forma de saber si el remitente es quien dice se.
- El **`timestamp`** nos permite detectar aquellas peticiones que se crearon hace más de 24 horas y descartarlas.
- Cada vez que recibimos una petición válida, almacenamos el **`nonce`** durante un tiempo. De esta forma garantizamos que una petición, sólo se reciba una única vez; esto es, todas aquellas peticiones cuyo ***`nonce`*** asociado ya haya sido registrado en el sistema, se descartarán. De esta manera, en el caso de que alguien ajeno repita el envío de una petición interceptada, esta petición será rechazada.
- El **`keyString`** se genera a partir de concatenar el ***`nonce`*** y el ***`timestamp`*** originales, generados en el código del cliente, antes de que la petición haya sido lanzada. De esta forma, si un interceptor opta por modificar el ***`nonce`*** o ***`timestamp`*** de la petición interceptada, ésta se rechazará porque el servidor generará el ***`keyString`*** a partir de las cabeceras modificadas y, por tanto, el hash resultante no coincidirá con la firma original recibida.

2.3.4.3 API expuesta

La securización de la API expuesta que captura los eventos enviados por el servicio externo, se ha implementado de manera similar a la mencionada en la sección anterior, pero en este caso siguiendo las especificaciones descritas en la documentación de *Mandrill*.

En este caso, el servicio externo firma las peticiones de manera parecida y la envía en la cabecera **`X-Mandrill-Signature`**. También, codificada en **Base64**, la genera a partir de una clave que únicamente deberían de conocer *Baboon* y *Mandrill*.

Para validar las peticiones, Se ha utilizado un middleware de terceros donde el proceso viene ya implementado, testeado y listo para integrarlo en aplicaciones *node.js*.

2.3.5 *Aggregation Framework* de MongoDB

Para la implementar la obtención de las analíticas e impacto de las campañas de email marketing se ha utilizado el *Aggregation Framework* de **MongoDB**. Esta herramienta nos permite realizar operaciones que van más allá de las típicas operaciones CRUD, operaciones capaces de procesar valores obtenidos a partir de múltiples documentos, llevar a cabo una variedad de operaciones sobre ellos y devolver un único resultado.

2.3.6 Testing

Durante la implementación de las distintas funcionalidades se han ido implementando de forma simultánea los tests correspondientes que verifican que cada componente desarrollado desempeña su función y, además, la desempeña correctamente.

Se ha intentado seguir el proceso de desarrollo de software *TDD*: *Test-Driven Development* (Desarrollo guiado por pruebas) en todo momento que ha sido posible.

Durante el *TDD* en primer lugar se crea el test. A continuación se ejecutan los tests, asumiendo que el test recién implementado va a fallar. Tras fallar, se procede a desarrollar la funcionalidad que haga que el test se resuelva. Cuando el test se ha resuelto, se procede a realizar las refactorizaciones pertinentes en el código. En último lugar, relanzamos los tests para comprobar que se resuelven tras la refactorización y volvemos a repetir de nuevo este mecanismo.

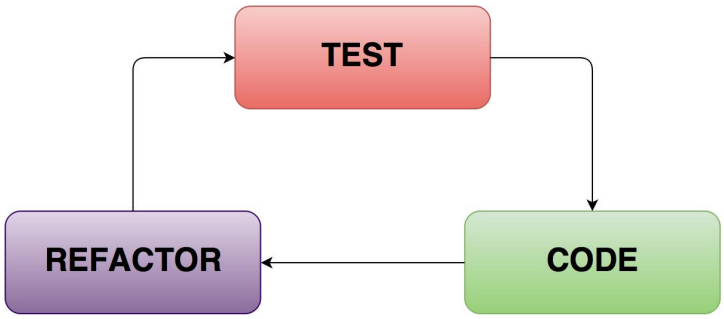


Figura 4. Test-Driven Development

En el **Anexo XI: Tests** se encuentran especificadas las distintas *suites* de tests que se han implementado para cada componente.

2.4 Despliegue

2.4.1 Contexto

En esta sección se recoge lo relacionado con el despliegue de la aplicación. En primer lugar se presenta la vista de despliegue de *Baboon*. A continuación, se describen los componentes software y sistema operativo necesarios para poder lanzarlo. Para finalizar, se explica todo el proceso de instalación de la aplicación donde se detalla la instalación de paquetes de servidor y componentes de cliente, la configuración por entornos y el empaquetado de la aplicación.

2.4.2 Vista de despliegue

Como puede observarse en la vista presentada a continuación, el despliegue de la aplicación se realiza sobre un mismo servidor, que se encargará de: servir el *frontend* de administración a los navegadores que lo soliciten y dar servicio a las aplicaciones que lo requieran a través de una API REST.

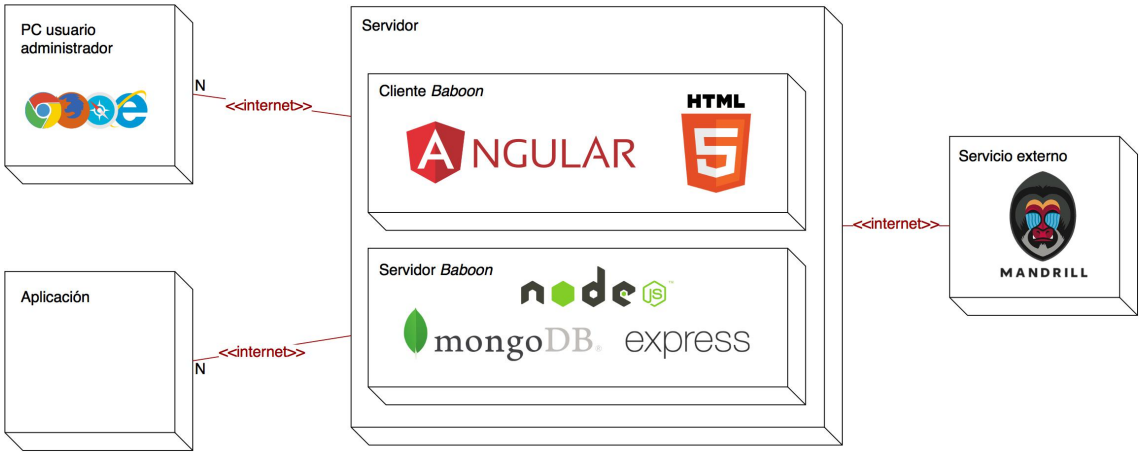


Figura 5. Vista de despliegue

2.4.3 Componentes software

Para poder utilizar el software *Baboon*, es necesario instalar las siguientes herramientas que se indican a continuación:

Nombre	Versión	Descripción
Node.js	0.12.*	La aplicación web se ejecuta a través de la plataforma ***Node.js***, construida sobre el motor de Javascript de Chrome y que permite construir aplicaciones de red rápidas y escalables.
MongoDB	3.0.*	
Bower	-	***Bower*** es un gestor de paquetes para la web y es necesario para instalar los distintos frameworks utilizados para construir la aplicación *Baboon*.

Grunt	-	***Grunt*** es un automatizador de tareas con el que se generan los scripts que hacen el empaquetado la aplicación *Baboon*.
Pm2[22]	-	***PM2*** es un gestor de procesos para aplicaciones *node.js* con un balanceador de carga integrado que permitirá sacar rendimiento a todos los cores del servidor.

2.4.4 Sistema operativo

Todos los componentes software utilizados se encuentran disponibles en los principales sistemas operativos, por lo que se puede instalar en el sistema que se prefiera de entre los siguientes: Linux, OSX, Sun Solaris o Windows.

Sin embargo se recomienda por rendimiento utilizar sistemas Linux que además es donde ha sido probado y desarrollado.

2.4.5 Instalación de BABOON

La aplicación será empaquetada en los servidores donde se vaya a realizar el despliegue, a partir del código fuente. Se utilizará el servidor de desarrollo/preproducción como lugar para descargar los paquetes necesarios, realizar las configuraciones pertinentes y hacer las pruebas. Una vez comprobado que todo es correcto se deberá mover la aplicación ya empaquetada al servidor de producción.

2.4.5.1 Instalación de paquetes de servidor

En el directorio raíz de la aplicación *Baboon* se encuentra el fichero **package.json** que contiene registradas las dependencias del lado del servidor. Para instalarlas, será necesario ejecutar sobre el directorio raíz el comando:

```
$ npm install
```

2.4.5.2 Instalación de componentes de cliente

En el directorio raíz de la aplicación *Baboon* se encuentra el fichero **bower.json** que contiene registradas las dependencias del lado del cliente. Para instalarlas, será necesario ejecutar sobre el directorio raíz el comando:

```
$ bower install
```

2.4.5.3 Configuración por entornos

La aplicación dispone de unos ficheros de configuración para parametrizar las funcionalidades de la misma, entre las que se encuentran:

- Datos del servidor
- Datos de acceso a la base de datos
- Datos de configuración del servidor de correo para envío de emails
- Datos de configuración para la conexión con *Mandrill*
- Datos de configuración del logger

[22] http://pm2.keymetrics.io

En el **Anexo XII: Configuración por entornos** se encuentran todos los detallados estos ficheros.

2.4.5.4 Empaquetado de la aplicación

Una vez instaladas todas las dependencias y configurados los entornos se puede proceder a empaquetar la aplicación.

Puesta en marcha en desarrollo

Una vez instaladas todas las dependencias se puede proceder a comprobar si la aplicación está correctamente instalada y configurada mediante la ejecución del siguiente comando:

```
$ grunt serve
```

Este comando arrancará la aplicación *Baboon* en modo *desarrollo* y permanecerá abierto en la terminal. Este es el modo de funcionamiento básico para hacer una primera comprobación del correcto funcionamiento.

Ejecución de los tests

Tras cada subida de una nueva versión es recomendable ejecutar los tests que se incluirán en la propia aplicación y que se encargarán de hacer las comprobaciones necesarias para garantizar que todo está funcionando. Esta tarea se ejecutará con el comando:

```
$ grunt test
```

Empaquetado

Una vez que se han hecho las comprobaciones pertinentes se puede proceder a empaquetar la aplicación para ser desplegada, lo que se hará a través del comando:

```
$ grunt dist
```

Esta tarea genera un directorio **`/dist`** dentro del proyecto con los componentes preparados para su instalación en modo servicio.

3 Gestión del proyecto

3.1 Metodología

En el desarrollo de este proyecto software se ha realizado una implementación incremental del producto dividida en iteraciones basada, pero no de manera estricta, en la metodología ágil Scrum.

Scrum es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto, donde se realizan entregas parciales y regulares del producto final priorizadas por el beneficio que aportan al receptor de proyecto. Por ello, esta metodología es especialmente la indicada en proyectos donde los requisitos son cambiantes o poco definidos, donde la innovación, la flexibilidad y la productividad son fundamentales.

Por lo tanto, debido a que el Trabajo Fin de Grado es un proyecto a nivel individual y Scrum un proceso aplicado a proyectos en equipo, no es posible ceñirse rigurosamente a la metodología. Es por ello que se ha optado por seguir algunas de las partes viables, dadas las circunstancias, y más significativas que el proceso Scrum ofrece. Entre ellas, cabe destacar las siguientes:

- Reuniones frecuentes (diarias o, en este caso mayormente semanales).
- Reuniones a nivel de *sprints*
    - **Sprint Planning:** reunión previa al lanzamiento del *sprint*, donde se realiza la planificación de la iteración que va a comenzar donde se seleccionan los requisitos más prioritarios a desarrollar en ella.
    - **Sprint retrospective:** reunión tras la finalización del *sprint*, cuyo objetivo es la mejora continua de la productividad y la calidad del producto que se está desarrollando realizando un análisis del trabajo desarrollado durante la iteración.
- Priorización de los requisitos o funcionalidades a la hora de desarrollar el producto. Se ha utilizado para esto, dos conceptos básicos de Scrum:
    - **Product Backlog:** (*lista de requisitos/objetivos priorizada* o *pila del producto*) representa la visión y expectativas del cliente, principal responsable de crear y gestionarla, respecto a los objetivos y entregas del producto. En esta lista se encuentran los requisitos ordenados según su prioridad.
    - **Sprint Backlog:** (*lista de tareas de la iteración*) en esta lista se encuentran los requisitos que han de implementarse durante el un determinado *sprint*.
- Asignación de los roles principales del proceso
    - **Equipo de desarrollo:** Es el equipo de desarrolladores multidisciplinario que de forma auto-organizada serán los encargados de desarrollar el producto. En este caso el equipo lo constituirá únicamente el propio alumno.
    - **Scrum master:** Persona conocedora del proceso Scrum que se encarga de orientar al equipo. En este caso, el director del proyecto.
    - **Dueño del producto:** La única persona autorizada para decidir sobre cuáles funcionalidades y características funcionales tendrá el producto. Es quien representa al cliente, usuarios del software y

todas aquellas partes interesadas en el producto. En este caso, un miembro de **10Labs**.

- **Scrum Taskboard:** (o *tablero o pizarra de tareas*) Tablero o pizarra que difunde el estado actual de la iteración, en el que cada tarea puede encontrarse en forma de tarjeta y donde se presentan los distintos estados posibles en los que éstas pueden encontrarse. Una tarjeta puede arrastrarse de un estado a otro conforme cambia de estado.

A pesar de haberse seguido un desarrollo incremental dividido en *sprints*, primeramente se ha optado por realizar un análisis y diseño general del problema a afrontar. Esto ha permitido adquirir una primera visión en detalle antes de entrar en materia.

Por último, cabe destacar también que al comienzo de cada una de las iteraciones se ha realizado un proceso de refinamiento de análisis y diseño, antes de comenzar la correspondiente implementación. Esto no tiene otro objetivo que la introducción de modificaciones o posibles mejoras en la solución, surgidas a lo largo del proceso de desarrollo llevado a cabo previamente.

3.2 Herramientas de gestión

A lo largo del proyecto se ha hecho uso de una serie de herramientas que han facilitado su gestión y, además, han permitido realizarla de una forma mucho más ágil. Las dos herramientas principales que se han utilizado en este caso son:

- **Trello[23]:** Tablero que permite la creación de listas y tarjetas, donde una lista se correspondería con el estado de una tarea y una tarjeta representaría a una tarea permitiendo arrastrar y soltar tarjetas entre las distintas listas y la ordenación de tarjetas dentro de una misma lista. A nivel de tarjeta, es posible añadir comentarios, crear checklists, añadir fechas de vencimiento, entre otras. Un tablero puede ser utilizado en solitario por una sola persona o por un equipo, todo en tiempo real y de forma muy rápida. Es una herramienta multiplataforma que funciona en todo tipo de dispositivos, ya sea de forma nativa (dispone de aplicaciones para iPhone, iPad, teléfonos, tabletas y relojes Android, y tabletas Kindle Fire) o en la web, a través de un navegador y con cualquier tamaño de pantalla.
- **Bitbucket[24]:** Solución *Git* orientada a equipos de profesionales. Se trata de un sistema de versiones distribuido que permite de forma sencilla la colaboración entre miembros de un equipo. Entre los servicios que ofrece, se incluyen la creación de repositorios privados y la securización de repositorios de forma granular a distintos niveles (proyecto, repositorio, rama).

Trello se ha utilizado durante todo el proceso de desarrollo como "tablero Scrum" donde se han creado una tarjeta por cada requisito o funcionalidad existente y, además, se han definido una serie de listas donde agrupar las tarjetas. Estas listas se detallan a continuación:

- **Product Backlog:** Representa la pila del producto.

[23] https://trello.com

[24] https://bitbucket.org

- **Sprint Backlog:** Representa la pila del *sprint* actual.
- **In progress:** Tareas (previamente en el **Sprint Backlog**) que se encuentran en desarrollo.
- **Ready for revision:** Tareas cuya implementación ha terminado y están listas para ser revisadas.
- **Closed:** Tareas cuya implementación ha terminado, han sido revisadas y funcionaban correctamente.

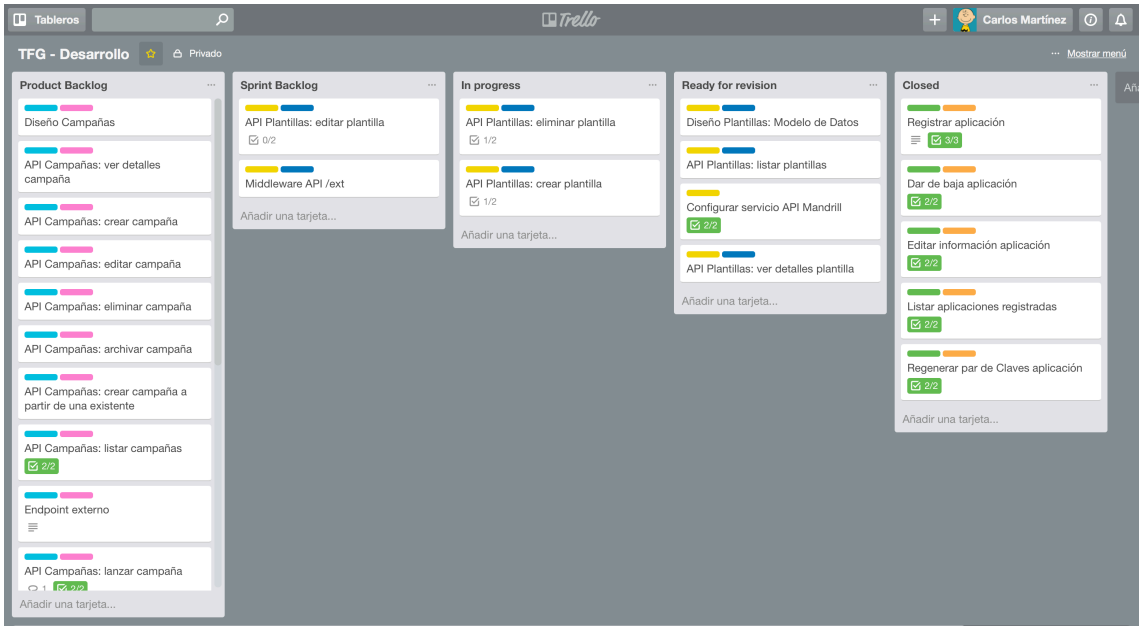


Figura 6. Herramienta Trello

Por otro lado, se ha utilizado **Bitbucket** como sistema de control de versiones. Inicialmente se creó un repositorio con el esqueleto del proyecto sobre el cual se ha ido trabajando y actualizando conforme nuevas funcionalidades han ido implementándose. Esta herramienta ofrece, asimismo, un registro de *issues* a nivel de repositorio que permite la gestión de posibles *bugs*, mejoras o tareas, priorizarlas e incluso asignarlas a cualquier miembro del equipo que tenga acceso al repositorio. Esta sistema de *tracking* que ofrece **Bitbucket** ha sido de gran utilidad, pues ha permitido registrar cualquier detalle que haya ido surgiendo a lo largo del desarrollo, permitiendo así su posterior en cualquier momento.

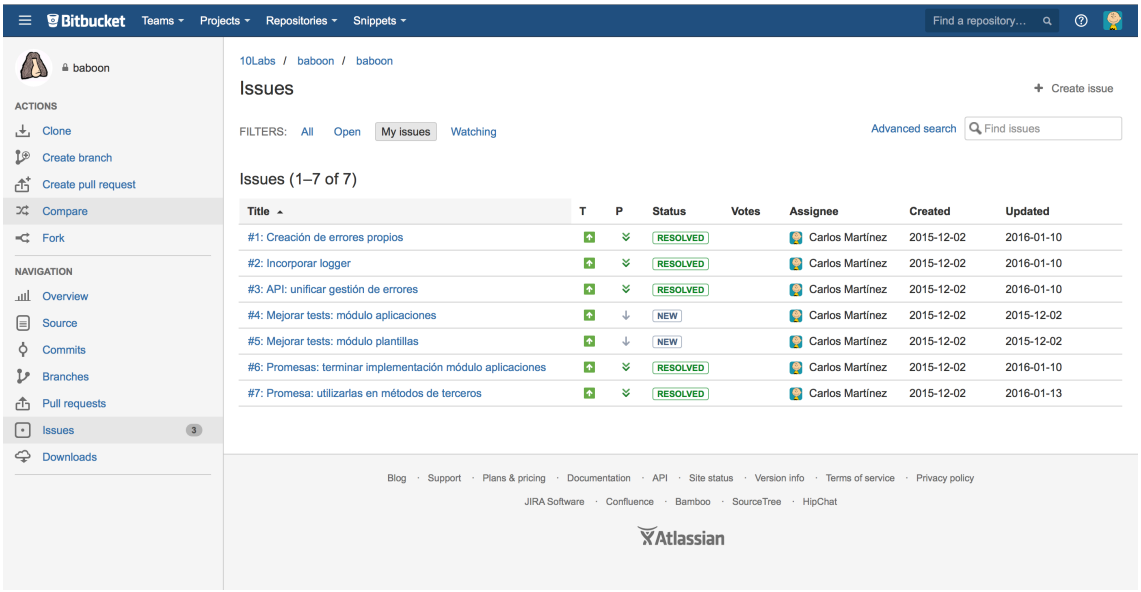


Figura 7. Herramienta Bitbucket

Por último, aparte de las dos herramientas citadas anteriormente, se han utilizado dos cuadernos clásicos cuadriculados de anillas para hacer anotaciones y recoger buena parte de la documentación a la vez que se ha ido desarrollando el trabajo:

- **Cuaderno de Ingeniería I:** Uno de ellos se ha utilizado principalmente para su uso en el día a día, hacer cualquier tipo de anotaciones, realizar todo tipo tipos de diseños con diagramas variados o simplemente plasmar ideas.
- **Cuaderno de Ingeniería II:** El otro se ha utilizado fundamentalmente para recoger documentación definitiva a partir de aquella registrada en el cuaderno del día a día, una vez extendida y clarificada. Además de esto, se ha utilizado una sección dentro de este cuaderno para llevar a cabo el registro de esfuerzos.

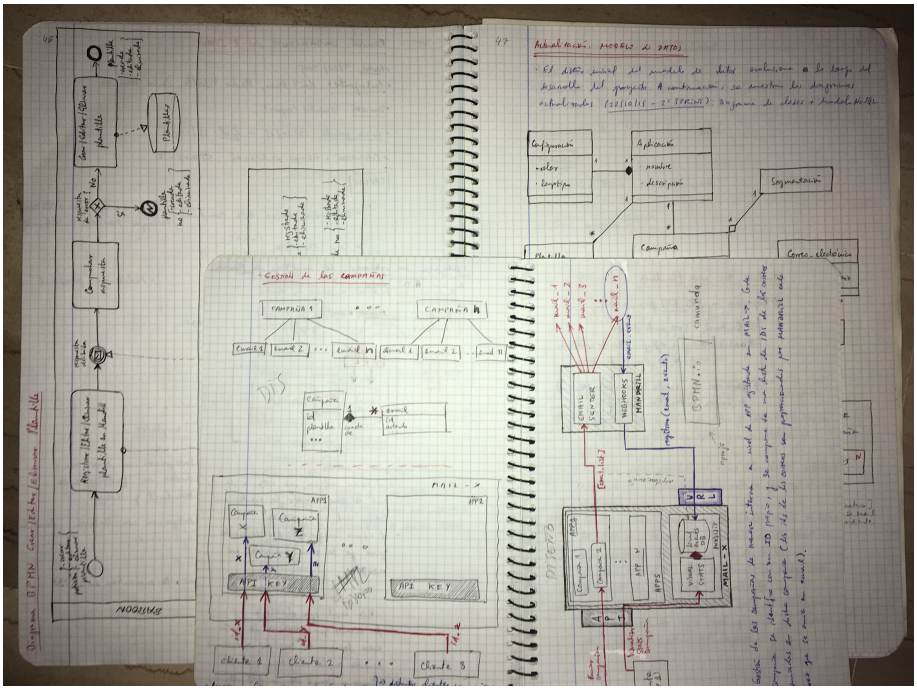


Figura 8. Cuadernos de ingeniería

3.3 Planificación

La planificación del trabajo realizado se ha llevado a cabo intentando seguir los principios de Scrum expuestos en la sección **3.1 Metodología**, realizando un desarrollo incremental del producto por iteraciones. De esta forma, inicialmente se optó por definir una lista priorizada de todas las funcionalidades (*Product backlog*) ordenadas de mayor a menor importancia, a partir de la cual acabaron definiéndose los *sprints* que finalmente fueron lanzados.

El mecanismo para la definición y planificación de cada iteración que se ha seguido ha sido siempre el mismo, y se detalla a continuación:

- Asignación de una fecha de inicio y fin al *sprint*, manteniendo siempre una duración aproximada de entre 2 y 3 semanas.
- Estimación aproximada del tiempo total que se le va a dedicar al *sprint*.
- Definición de la lista de tareas de la iteración (*Sprint backlog*).
- Realización de un desglose de las funcionalidades, en subtareas.
- Estimación aproximada del tiempo que costará implementar cada subtarea.
- Obtención del tiempo total estimado que costará implementar todas las subtareas:
    - **si** *tiempo_estimado_tareas* > *tiempo_dedicación_sprint* → se mueven tareas del *Sprint Backlog* de vuelta al *Product Backlog*.
- Se procede a llevar a cabo el lanzamiento del *sprint.*

Y de tal modo, tras seguir este proceso, el desarrollo del producto acabó realizándose a lo largo 4 iteraciones cuya planificación se presenta a continuación de forma más precisa.

3.3.1 Planificación SPRINT-1

En primera instancia se llevó a cabo una planificación y lanzamiento de la primera iteración, pero poco después de su lanzamiento se optó por desarrollar una replanificación completa del *sprint* y por redefinir la pila del producto de nuevo.

El motivo principal fue que la funcionalidad que había que comenzar desarrollando, a pesar de ser la más importante del producto, era de las más complejas. Fue por ello, y que el alumno no poseía por aquel momento la experiencia y soltura suficiente en algunas de las tecnologías empleadas, que se decidió replantear de nuevo la planificación.

De este modo se procedió a efectuar una nueva replanificación tomando como prioridad la dificultad de las tareas frente a la importancia de dicha funcionalidad en base al producto final, teniendo que sacrificar así uno de los principios fundamentales de las metodologías ágiles: conseguir tras la conclusión de cada iteración un producto mínimo viable, algo que no fue posible conseguir hasta el final de la tercera.

Para más detalle sobre planificación y estimaciones iniciales de esta iteración, consultar **Anexo XIII: Planificación inicial SPRINT-1**.

Para más detalle sobre la replanificación y estimaciones llevadas a cabo en esta iteración, consultar **Anexo XIV: Replanificación SPRINT-1**.

3.3.2 Planificación SPRINT-2

La planificación llevada a cabo en la segunda iteración del desarrollo ha resultado ser la más imprecisa de todas las realizadas, pues se efectuaron unas estimaciones de 2 semanas y el *sprint* acabó extendiéndose hasta un total de entre 4 y 5 semanas finalmente.

El motivo de este desfase vuelve a tener algo en común con la replanificación realizada durante la primera iteración: la falta de experiencia en aquel momento por parte del alumno en alguna de las tecnologías, que en aquel momento produjo esta demora en la finalización del *sprint*.

Para más detalle sobre planificación y estimaciones de esta iteración, consultar **Anexo XV: Planificación SPRINT-2**.

3.3.2 Planificación SPRINT-3

Durante la tercera iteración se llevó a cabo la implementación de la parte más crítica e importante del producto, aquella que en principio se planteó desarrollar durante el primer *sprint*.

A diferencia que en los comienzos, esta vez consiguieron abordarse las tareas sin ningún tipo de contratiempo; la experiencia adquirida a lo largo de las dos primeras iteraciones permitieron concluir el *sprint* sin dificultades, dentro del plazo estipulado.

Para más detalle sobre planificación y estimaciones de esta iteración, consultar **Anexo XVI: Planificación SPRINT-3**.

3.3.2 Planificación SPRINT-4

El cuarto y último *sprint*, se empleó para realizar los últimos ajustes e implementar la funcionalidad restante del producto.

En la misma dinámica que el tercero, no presentó ningún tipo de contratiempo y fue posible darlo por concluido dentro de lo estimado; y a diferencia de los demás, fue la iteración más corta, con una duración de 2 semanas frente a las resto.

Para más detalle sobre planificación y estimaciones de esta iteración, consultar **Anexo XVII: Planificación SPRINT-4**.

3.4 Esfuerzos

Como se ha comentado anteriormente en la sección **3.2 Herramientas de gestión**, se ha utilizado uno de los cuadernos de ingeniería para realizar el registro de los esfuerzos que se ha hecho detalladamente por fecha y categoría. A continuación, se explican las categorías bajo las cuales éstos se han catalogado:

- **Planificación:** Todo tipo de reuniones llevadas a cabo, incluyendo el lanzamiento del proyecto y los *sprints*, las reuniones diarias y otras reuniones para solucionar dudas concretas, así como las tareas de gestión y configuración del proyecto.
- **Análisis:** Todas aquellas tareas de análisis del proyecto que se han hecho de carácter general en la fase inicial del proyecto y aquellas a nivel de *sprint.*
- **Diseño:** Todas aquellas tareas de diseño del proyecto que se han hecho de carácter general en la fase inicial del proyecto y aquellas a nivel de *sprint.*

- **Implementación:** Todas las tareas de desarrollo de software que se han llevado a cabo, concretamente la implementación de las funcionalidades y sus correspondientes tests.
- **Documentación:** Todo tipo de tarea relacionada con la documentación del proyecto, concretamente la transcripción de toda aquella documentación importante desarrollada al cuaderno de ingeniería y el desarrollo de la memoria del Trabajo Fin de Grado.
- **Formación:** Tareas relacionadas con la ampliación del conocimiento en materias necesarias para la resolución del problema como la lectura de libros y artículos sobre *MongoDB*, y la utilización de Promesas y testing en Javascript, así como la asistencia a un curso de aproximadamente 35 horas de *AngularJS* impartido por Raúl Novoa, el propio director del Trabajo Fin de Grado

En la siguiente tabla se recogen todos los esfuerzos registrados desde el primer hasta el último día, agrupados por categoría, y la suma total de todos ellos:

Planificación	42 horas
Análisis	32 horas
Diseño	27 horas
Implementación	228 horas
Documentación	110 horas
Formación	55 horas
TOTAL	**494 horas**

4 Conclusiones

4.1 Resultados

El resultado obtenido no trata únicamente de un producto que funciona y que , por tanto, permite gestionar campañas de email marketing; no es solo eso. Nos encontramos ante un Proyecto de Software completo, para el cual se ha desarrollado previamente un detenido análisis, generado documentación asociada y testeado la solución de forma exhaustiva.

Con *Baboon* se ha conseguido ofrecer un servicio a través de la nube para toda aplicación o futura aplicación que lo necesite y quiera hacer uso de él, de forma sencilla y además, segura.

Estamos, por tanto, frente la primera versión de un producto real del cual se va a hacer uso en un entorno de producción en un futuro próximo y que no resultaría nada complejo seguir extendiéndolo, añadiendo nuevas funcionalidades o completando las ya existentes.

4.2 Lecciones aprendidas

Este Trabajo Fin de Grado ha permitido profundizar y ampliar los conocimientos del alumno especialmente en materias relacionadas con la Ingeniería de Software y la Ingeniería Web.

Además, también le ha dado la oportunidad de aprender algunas de las tecnologías más actuales, las cuales conocía, pero no había llegado a utilizar todas ellas ni tampoco de forma tan precisa.

Por otro lado, el estudiante ha aprendido también utilizar y diseñar modelos de datos de manera totalmente distinta a la que estaba acostumbrado hasta ahora (el modelo relacional) pues *MongoDB*, el sistema de base de datos empleada, es un sistema NoSQL orientado a documentos.

En adición a todo lo anterior, las naturaleza del trabajo ha conseguido que el alumno fuese todavía más consciente de la importancia del papel que tienen los tests y el aplicar buenas prácticas durante el proceso de desarrollo de un proyecto

4.3 Conclusión personal

La experiencia a lo largo de todo el proceso de desarrollo del Trabajo Fin de Grado en **10Labs** ha sido muy positiva. Desde el primer día me he sentido muy a gusto allí y les agradezco la oportunidad que me han dado de aprender con y de ellos.

El proyecto en su totalidad ha cumplido con todas mis expectativas, pues me ha permitido aplicar buena parte de los conocimientos que he adquirido todos estos años en la Universidad y, además, ampliarlos.

Desde el principio, el principal objetivo del TFG para mi ha sido siempre conseguir aprender lo máximo posible y, a su vez, adquirir cierta experiencia en determinadas competencias, cosa que me ha permitido el desarrollar este trabajo.

Finalmente, por todo esto, he de decir que me siento muy satisfecho por todo el esfuerzo invertido, el trabajo realizado y el resultado obtenido a raíz de ello.

5 Bibliografía

Javascript: The Good Parts, *O'Reilly Media*: este libro ha sido de gran utilidad par conocer más en profundidad los detalles del lenguaje de programación Javascript, utilizado durante el desarrollo del proyecto.

Node: Up and Running, *O'Reilly Media*: este libro ha sido de gran utilidad para comprender y conocer mejor la arquitectura de *node.js* y su modelo de programación basada en eventos.

Documenting Software Architectures: Views and Beyond, *2nd Edition*: este libro ha sido de utilidad a la hora de realizar los diseños de componentes y vistas.

We have a problem with promises: Muy buen artículo que profundiza en la utilización de promesas Javascript y su entendimiento. Ha sido de gran utilidad como material formativo en este tema. *https://pouchdb.com/2015/05/18/we-have-a-problem-with-promises.html*

Signing and Authenticating REST Requests: Documentación de *Amazon* donde estan explicados los mecanismos que utilizan para securizar sus APIs Web mediante *API Keys*. A servido de inspiración, entre otras cosas, para desarrollar el middleware de securización de la API externa. *http://docs.aws.amazon.com/AmazonS3/latest/dev/RESTAuthentication.html#UsingTemporarySecurityCredentials*

Por otra parte, se ha consultado también la documentación oficial de cada una de las tecnologías y herramientas utilizadas durante todo el proceso, las cuales aparecen a pie de página en la memoria conforme éstas son nombradas por primera vez.

Anexo I: Casos de Uso

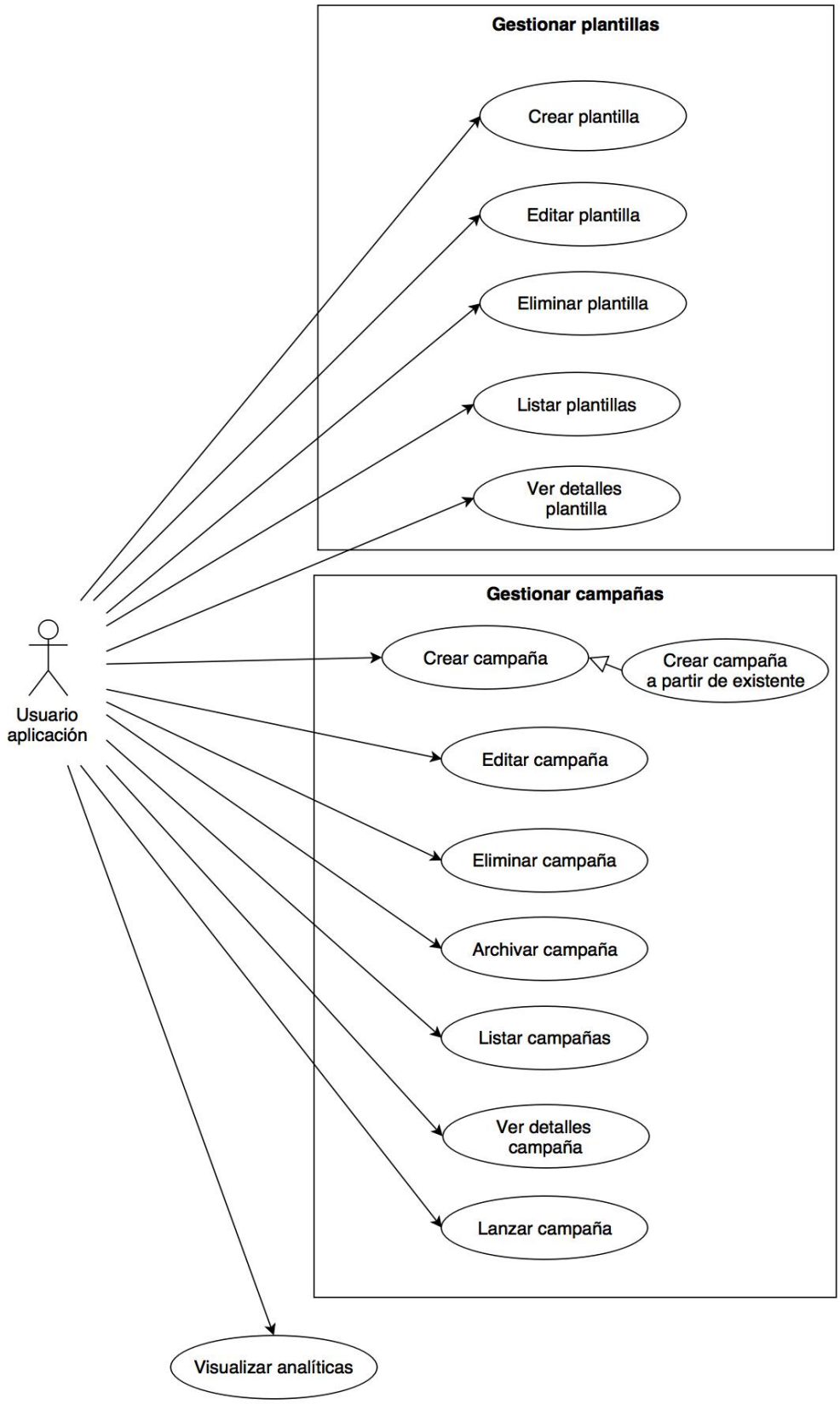


Figura 9. Casos de Uso detallados

A continuación, se exponen detalladamente los Casos de Uso del sistema:

Registrar Aplicación

Este caso de uso lo utiliza un usuario administrador para registrar aplicaciones que van a hacer uso del sistema.

Precondición: Ninguna.

Postcondición: Queda registrada una nueva aplicación en el sistema que, a partir de ese momento, puede hacer uso de toda la funcionalidad que éste ofrece.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario administrador solicita el registro de una aplicación en el sistema.

2. El sistema le proporciona el registro y el usuario administrador introduce los datos correspondientes.

3. El sistema valida los datos introducidos, la aplicación se registra correctamente y termina el caso de uso.

Flujo de eventos alternativo:

3. Los datos introducidos no son válidos, el sistema se lo notifica al usuario administrador y vuelve a solicitar de nuevo los datos del registro.

Flujo de eventos alternativo:

2. La aplicación le proporciona el registro y el usuario administrador lo cancela, terminando así el caso de uso.

Editar aplicación

Este caso de uso lo utiliza un usuario administrador para introducir, modificar o eliminar la información de una aplicación registrada en el sistema.

Precondición: Existen aplicaciones registradas en el sistema.

Postcondición: Se ha modificado la información de una de las aplicaciones registradas en el sistema.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario administrador solicita editar la información de una de las aplicaciones registradas en el sistema.

2. El sistema le proporciona la edición y el usuario administrador introduce las modificaciones correspondientes.

3. El sistema valida las modificaciones introducidas, actualiza la información de la aplicación correspondiente y termina el caso de uso.

Flujo de eventos alternativo:

3. Las modificaciones introducidas no son válidas, el sistema se lo notifica al usuario administrador y vuelve a solicitar de nuevo que introduzca las modificaciones.

Flujo de eventos alternativo:

2. El sistema le proporciona la edición y el usuario administrador la cancela, terminando así el caso de uso.

Desactivar aplicación

Este caso de uso lo utiliza un usuario administrador para desactivar una aplicación registrada en el sistema, restringiendo así su uso del sistema.

Precondición: Existen aplicaciones registradas y no desactivadas en el sistema.

Postcondición: Se ha desactivado una de las aplicaciones registradas en el sistema y a partir de ese momento no podrá hacer uso del sistema.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario administrador solicita desactivar una aplicación registrada en el sistema.

2. El sistema le proporciona la desactivación de la aplicación, el usuario la confirma y termina el caso de uso.

Activar aplicación

Este caso de uso lo utiliza un usuario administrador para activar una aplicación registrada en el sistema, permitiéndole de nuevo hacer uso del sistema.

Precondición: Existen aplicaciones registradas y desactivadas en el sistema.

Postcondición: Se ha activado una de las aplicaciones registradas en el sistema y a partir de ese momento volverá a poder hacer uso del sistema.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario administrador solicita activar una aplicación registrada en el sistema.

2. El sistema le proporciona la activación de la aplicación, el usuario la confirma y termina el caso de uso.

Crear plantilla

Este caso de uso lo utiliza un usuario aplicación para crear una plantilla de correo electrónico.

Precondición: Ninguna.

Postcondición: Se ha creado una nueva plantilla de correo electrónico.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario aplicación solicita la creación de una plantilla de correo electrónico.

2. El sistema proporciona la creación de una plantilla y el usuario aplicación introduce los datos correspondientes.

3. El sistema valida los datos introducidos, la plantilla se crea correctamente y termina el caso de uso.

Flujo de eventos alternativo:

3. Los datos introducidos no son válidos, el sistema se lo notifica al usuario aplicación y vuelve a solicitar de nuevo los datos de creación de una plantilla.

Flujo de eventos alternativo:

2. El sistema le proporciona la creación de una plantilla y el usuario aplicación lo cancela, terminando así el caso de uso.

Editar plantilla

Este caso de uso lo utiliza un usuario aplicación para introducir, modificar o eliminar la información de una de sus plantillas existentes en el sistema.

Precondición: Existen plantillas en el sistema, propiedad del usuario aplicación.

Postcondición: Se ha modificado la información de una de las plantillas del usuario aplicación.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario aplicación solicita editar una plantilla de correo electrónico.

2. El sistema le proporciona la edición y el usuario aplicación introduce las modificaciones correspondientes.

3. El sistema valida las modificaciones introducidas, actualiza la información de la plantilla y termina el caso de uso.

Flujo de eventos alternativo:

3. Las modificaciones introducidas no son válidas, el sistema se lo notifica al usuario aplicación y vuelve a solicitar de nuevo que introduzca las modificaciones.

Flujo de eventos alternativo:

2. El sistema le proporciona la edición y el usuario aplicación la cancela, terminando así el caso de uso.

Eliminar plantilla

Este caso de uso lo utiliza un usuario aplicación para eliminar una de sus plantillas existentes en el sistema.

Precondición: Existen plantillas en el sistema, propiedad del usuario aplicación.

Postcondición: Se ha eliminado una de las plantillas del usuario aplicación.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario aplicación solicita eliminar una plantilla de correo electrónico.

2. El sistema solicita al usuario aplicación la confirmación de la eliminación y éste la confirma.

3. El sistema elimina la plantilla seleccionada por el usuario y termina el caso de uso.

Flujo de eventos alternativo:

3. El sistema solicita al usuario aplicación la confirmación de la eliminación, éste la cancela y termina el caso de uso.

Listar plantillas

Este caso de uso lo utiliza un usuario aplicación para listar sus plantillas.

Precondición: Ninguna.

Postcondición: Se han listado las plantillas del usuario aplicación.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario aplicación solicita obtener el listado de sus plantillas existentes.

2. El sistema le proporciona el listado de plantillas al usuario aplicación y termina el caso de uso.

Ver detalles plantilla

Este caso de uso lo utiliza el usuario aplicación para visualizar la información de una de sus plantillas.

Precondición: Existen plantillas en el sistema, propiedad del usuario aplicación.

Postcondición: Se ha visualizado la información de una plantilla del usuario aplicación.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario aplicación solicita visualizar la información de una de sus plantillas existentes.

2. El sistema le proporciona los detalles de la plantilla al usuario aplicación y termina el caso de uso.

Crear campaña

Este caso de uso lo utiliza el usuario aplicación para crear una campaña de email marketing.

Precondición: Existen plantillas en el sistema, propiedad del usuario aplicación..

Postcondición: Se ha creado una nueva campaña de email marketing.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario aplicación solicita la creación de una campaña de email marketing.

2. El sistema proporciona la creación de una campaña de email marketing, el usuario aplicación escoge la plantilla que usará la campaña e introduce los datos correspondientes.

3. El sistema valida los datos introducidos, la campaña se crea correctamente y termina el caso de uso.

Flujo de eventos alternativo:

3. Los datos introducidos no son válidos, el sistema se lo notifica al usuario aplicación y vuelve a solicitar de nuevo los datos de creación de una campaña de email marketing.

Flujo de eventos alternativo:

2. El sistema le proporciona la creación de una campaña de email marketing y el usuario aplicación lo cancela, terminando así el caso de uso.

Crear campaña a partir de existente

Este caso de uso lo utiliza el usuario aplicación para crear una campaña de email marketing a partir de otra de sus campañas ya enviadas.

Precondición: Existen campañas de email marketing enviadas, propiedad del usuario aplicación.

Postcondición: Se ha creado una nueva campaña de email marketing a partir de una campaña existente.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario aplicación solicita la creación de una campaña de email marketing a partir de otra campaña ya existente.

2. El usuario aplicación escoge una de sus campañas ya enviadas a partir de la cual creará la nueva campaña.

3. El sistema proporciona la edición de la información y el usuario introduce los cambios que considere oportunos en la nueva campaña.

4. El sistema valida los datos introducidos, la campaña se crea correctamente y termina el caso de uso.

Flujo de eventos alternativo:

4. Los datos introducidos no son válidos, el sistema se lo notifica al usuario aplicación y vuelve a solicitar de nuevo la edición de la información en la nueva campaña de email marketing.

Flujo de eventos alternativo:

3. El sistema le proporciona la edición de la información y el usuario aplicación la cancela, terminando así el caso de uso.

Editar campaña

Este caso de uso lo utiliza un usuario aplicación para introducir, modificar o eliminar la información de una de sus campañas de email marketing no enviadas, existente en el sistema.

Precondición: Existen campañas de email marketing no enviadas, propiedad del usuario aplicación.

Postcondición: Se ha modificado la información de una de las campañas de email marketing del usuario aplicación.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario aplicación solicita editar una campaña de email marketing.

2. El sistema le proporciona la edición y el usuario aplicación introduce las modificaciones correspondientes.

3. El sistema valida las modificaciones introducidas, actualiza la información de la campaña y termina el caso de uso.

Flujo de eventos alternativo:

3. Las modificaciones introducidas no son válidas, el sistema se lo notifica al usuario aplicación y vuelve a solicitar de nuevo que introduzca las modificaciones.

Flujo de eventos alternativo:

2. El sistema le proporciona la edición y el usuario aplicación la cancela, terminando así el caso de uso.

Eliminar campaña

Este caso de uso lo utiliza un usuario aplicación para eliminar una de sus campañas de email marketing no enviadas, existentes en el sistema.

Precondición: Existen campañas de email marketing no enviadas, propiedad del usuario aplicación.

Postcondición: Se ha eliminado una campaña de email marketing del usuario aplicación.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario aplicación solicita eliminar una campaña de email marketing.

2. El sistema solicita al usuario aplicación la confirmación de la eliminación y éste la confirma.

3. El sistema elimina la plantilla seleccionada por el usuario y termina el caso de uso.

Flujo de eventos alternativo:

3. El sistema solicita al usuario aplicación la confirmación de la eliminación, éste la cancela y termina el caso de uso.

Archivar campaña

El caso de uso lo utiliza un usuario aplicación para archivar una de sus campañas de email marketing ya enviadas.

Precondición: Existen campañas de email marketing enviadas, propiedad del usuario aplicación.

Postcondición: Se ha archivado una campaña de email marketing del usuario aplicación.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario aplicación solicita archivar nua campaña de email marketing.

2. El sistema le proporciona el archivado de la aplicación, el usuario lo confirma y termina el caso de uso.

Listar campañas

Este caso de uso lo utiliza un usuario aplicación para listar sus campañas de email marketing.

Precondición: Ninguna.

Postcondición: Se han listado las campañas del usuario aplicación.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario aplicación solicita obtener el listado de sus campañas de email marketing existentes.

2. El sistema le proporciona el listado de campañas al usuario aplicación y termina el caso de uso.

Ver detalles campaña

Este caso de uso lo utiliza el usuario aplicación para visualizar la información de una de sus campañas de email marketing.

Precondición: Existen campañas en el sistema, propiedad del usuario aplicación.

Postcondición: Se ha visualizado la información de una campaña del usuario aplicación.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario aplicación solicita visualizar la información de una de sus campañas de email marketing existentes.

2. El sistema le proporciona los detalles de la campaña al usuario aplicación y termina el caso de uso.

Lanzar campaña

Este caso de uso lo utiliza el usuario aplicación para lanzar una de sus campañas de email marketing no enviadas.

Precondición: Existen campañas de email marketing no enviadas, propiedad del usuario aplicación.

Postcondición: Se ha lanzado una campaña de email marketing del usuario aplicación.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario aplicación solicita el lanzamiento de una de sus campañas.

2. El sistema le proporciona el lanzamiento de la campaña, el usuario aplicación lo confirma y termina el caso de uso.

Visualizar analíticas

Este caso de uso lo utiliza el usuario aplicación para visualizar las analíticas básicas de una de sus campañas de email marketing enviadas, como el número de correos enviados, rechazados, entregados, abiertos, etc.

Precondición: Existen campañas de email marketing enviadas, propiedad del usuario aplicación.

Postcondición: Se muestran las analíticas básicas de una campaña de email marketing.

Flujo de eventos principal:

1. El caso de uso comienza cuando el usuario aplicación solicita la visualización de las analíticas básicas de una campaña.

2. El sistema le proporciona las analíticas básicas de la campaña al usuario aplicación y termina el caso de uso.

Anexo II: Requisitos funcionales

A continuación se exponen detalladamente los requisitos funcionales del sistema, organizados por usuario: *usuario administrador* y *usuario aplicación*.

Usuario administrador

Los requisitos funcionales del sistema correspondientes al usuario administrador se detallan a continuación bajo el código **RF-Adm_XX** (donde **XX** especifica el número de requisito).

RF-Adm_01: El usuario administrador deberá ser capaz de registrar aplicaciones especificando

un nombre de aplicación,

una descripción y

un logotipo

las cuales tras finalizar el registro tendrán asociadas un par de claves

API Key y

API Secret

que les permitirán utilizar el sistema a través de un API Web.

RF-Adm_02: El usuario administrador podrá modificar

el nombre,

la descripción y

el logotipo

de cualquier aplicación registrada en el sistema.

RF-Adm_03: El usuario administrador podrá obtener un listado de las aplicaciones registradas en el sistema.

RF-Adm_04: El usuario administrador podrá generar un nuevo par de claves

API Key y

API Secret

asociado a cualquier aplicación registrada en el sistema.

RF-Adm_05: El usuario administrador podrá denegar la utilización del sistema a cualquier aplicación registrada cuando lo desee.

RF-Adm_06: El usuario administrador podrá permitir de nuevo la utilización del sistema a cualquier aplicación registrada a la que le haya denegado la utilización del sistema previamente.

Usuario aplicación

Los requisitos funcionales del sistema correspondientes al usuario aplicación se detallan a continuación bajo el código **RF-Usr_XX** (donde **XX** especifica el número de requisito).

RF-Usr_01: El usuario aplicación deberá ser capaz de crear plantillas de correo electrónico personalizadas especificando

un nombre de plantilla,

una descripción y

el contenido HTML de la plantilla,

del cual deberá especificar, si los hay,

los bloques de contenido que lo componen mediante

un nombre de bloque de contenido y

una descripción.

RF-Usr_02: El usuario aplicación podrá modificar

el nombre,

la descripción y

el contenido HTML,

del cual podrá modificar, si los hay,

el nombre y

la descripción

de los bloques de contenido que lo componen

de cualquiera de sus plantillas.

RF-Usr_03: El usuario aplicación podrá eliminar cualquiera de sus plantillas.

RF-Usr_04: El usuario aplicación podrá obtener un listado de sus plantillas.

RF-Usr_05: El usuario aplicación deberá ser capaz de crear campañas de email marketing especificando

un nombre de campaña,

una descripción,

un asunto de correo electrónico,

una dirección de correo electrónico remitente,

un nombre asociado a la dirección de correo electrónico remitente,

unos grupos de segmentación a los que irá destinada y

una plantilla de correo electrónico,

de la cual deberá especificar, si ésta dispone de alguno,

el contenido asociado a sus bloques de contenido.

las cuales tras finalizar su creación se encontrarán en estado *borrador*.

RF-Usr_05: El usuario aplicación podrá modificar

el nombre,

la descripción,

el asunto de correo electrónico,

la dirección de correo electrónico remitente,

el nombre asociado a la dirección de correo electrónico remitente,

los grupos de segmentación a los que irá destinada y

la plantilla de correo electrónico que utilizará

de la cual podrá modificar, si esta dispone de alguno,

el contenido asociado a sus bloques de contenido

de cualquiera de sus campañas de email marketing que se encuentren en estado *borrador*.

RF-Usr_06: El usuario aplicación podrá eliminar cualquiera de sus campañas de email marketing que se encuentren en estado *borrador*.

RF-Usr_07: El usuario aplicación podrá enviar sus campañas de email marketing que se encuentren en estado *borrador*, las cuales tras ser enviadas pasarán a encontrarse en estado *enviado*.

RF-Usr_08: El usuario aplicación podrá archivar sus campañas de email marketing que se encuentren en estado *enviado*, las cuales tras ser archivadas pasarán a encontrarse en estado *archivado*.

RF-Usr_09: El usuario aplicación podrá crear nuevas campañas de email marketing a partir de alguna de sus campañas ya existentes que se encuentren en estado *enviado* o *archivado*, las cuales poseerán

el nombre,

la descripción,

el asunto de correo electrónico,

la dirección de correo electrónico remitente,

el nombre asociado a la dirección de correo electrónico remitente,

los grupos de segmentación a los que irá destinada y

la plantilla de correo electrónico

y el contenido asociado a sus bloques de contenido

de la campaña elegida y tras finalizar su creación se encontrarán en estado *borrador*.

RF-Usr_10: El usuario aplicación podrá obtener un listado de sus campañas de email marketing.

RF-Usr_11: El usuario aplicación podrá visualizar el impacto de sus campañas, concretamente

el número total de correos electrónicos que constituyen la campaña,

el número de correos electrónicos pendientes de envío,

el número de correos electrónicos que no han podido ser entregados y

el número de correos electrónicos enviados correctamente,

de los cuales podrá visualizar

el número de correos electrónicos abiertos y

el número de correos electrónicos marcados como spam.

Anexo III: Requisitos no funcionales

Los requisitos no funcionales del sistema se detallan a continuación bajo el código **RNF- XX** (donde **XX** especifica el número de requisito).

RNF- 01: La administración básica del sistema, que se corresponde con

el registro, edición y listado de aplicaciones,

permitir/denegar a una aplicación registrada utilizar el sistema,

la regeneración del par de claves de acceso al sistema de una aplicación

se realizará a través de una aplicación web.

RNF- 02: La utilización del sistema por parte de las aplicaciones registradas, que se corresponde con

la creación, edición, eliminado y listado de plantillas,

la creación, edición, eliminado, listado, archivado y envío de campañas y

la visualización de analíticas básicas de una campaña

se realizará a través de un API Web.

RNF- 03: La utilización del API Web requerirá de autenticación mediante el par de claves

API Key y

API Secret

generadas por el sistema tras el registro de una aplicación.

Anexo IV: BPMN Crear plantilla

A continuación se detalla el flujo que seguiría la funcionalidad *Crear Plantilla* a través de un diagrama BPMN:

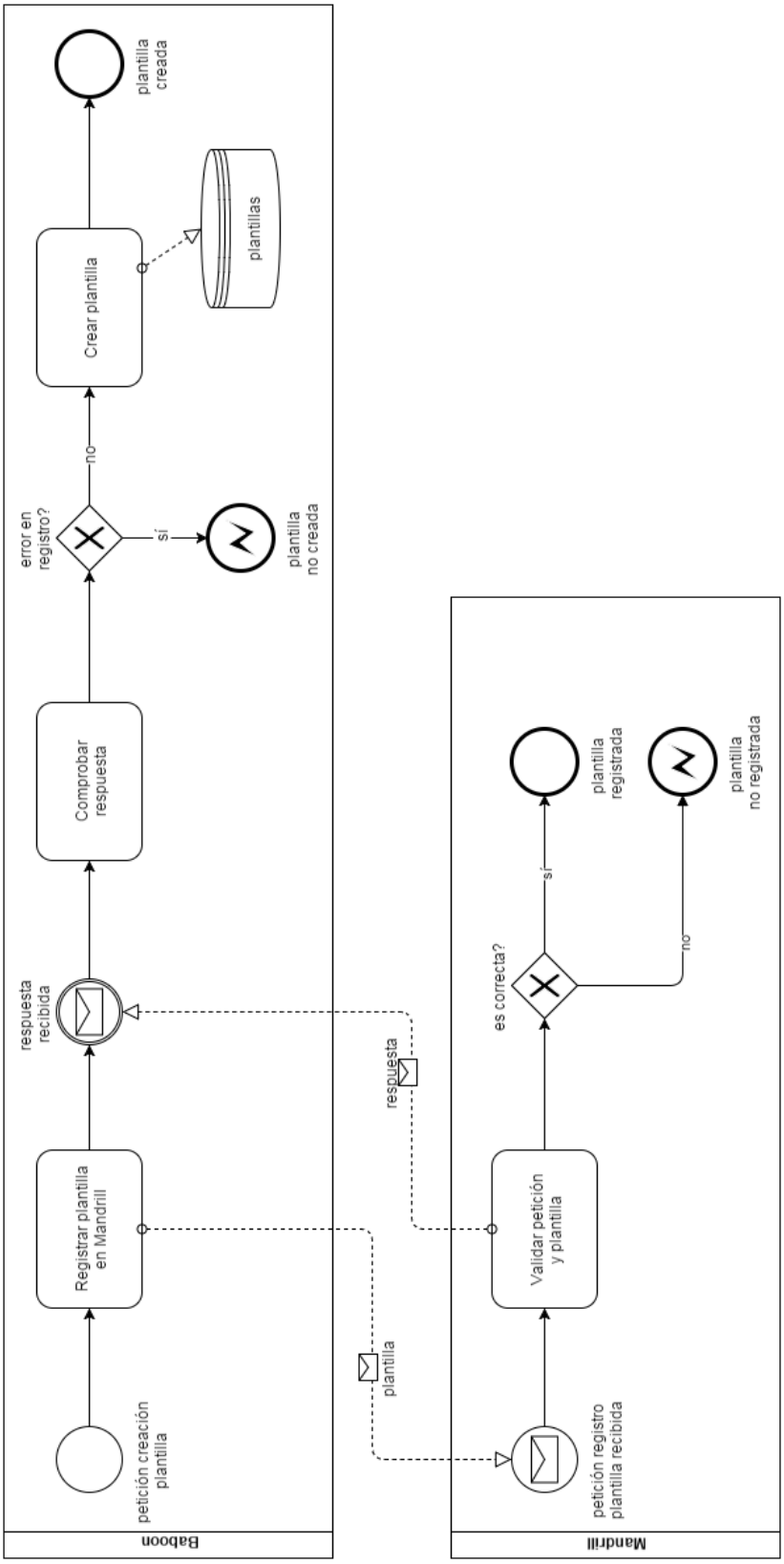


Figura 10. BPMN Crear Plantilla

Anexo V: BPMN Envío correo electrónico

A continuación se detalla el flujo de ejecución asociado al envío de un correo electrónico a través de un diagrama BPMN:

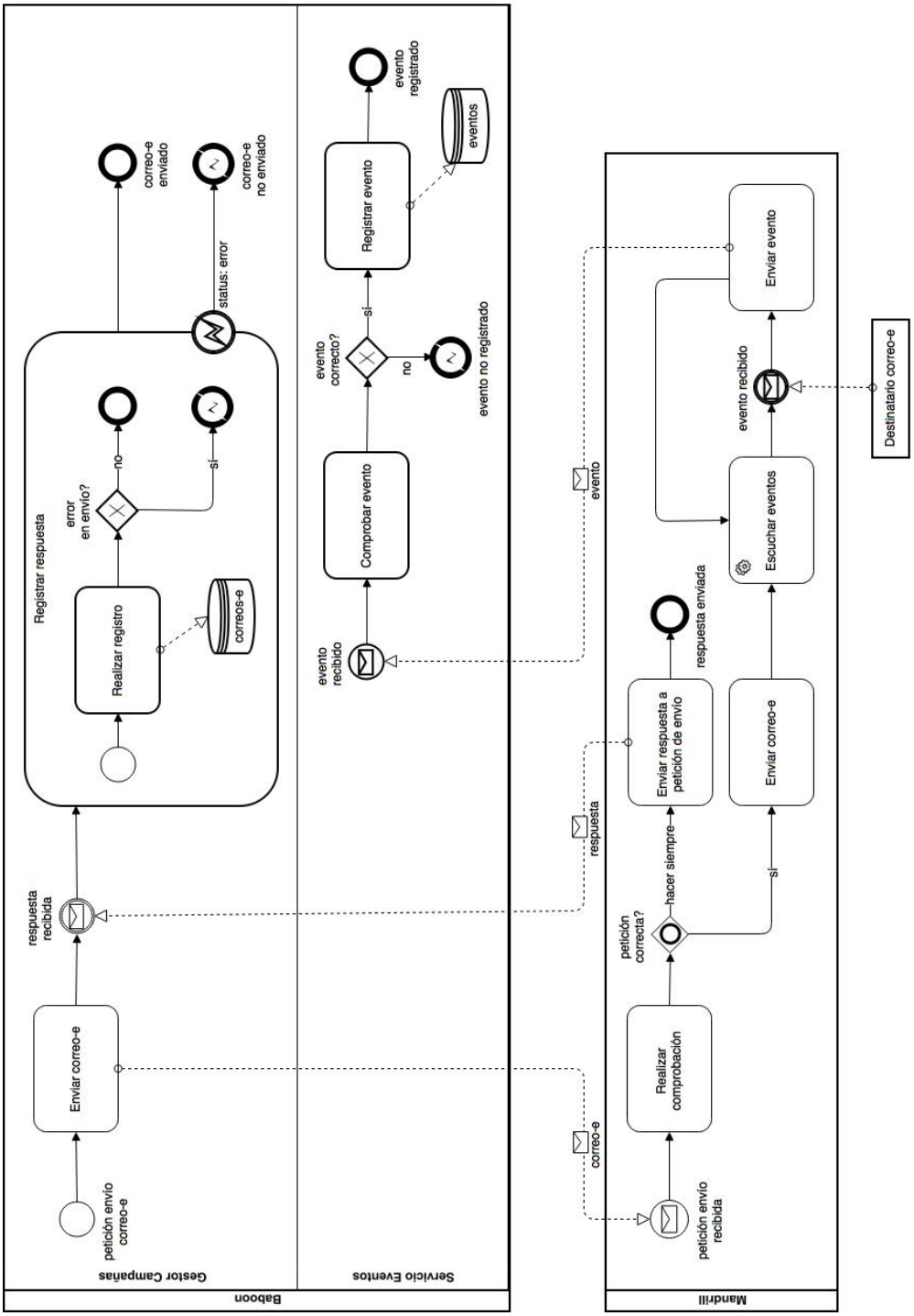


Figura 11. BPMN Flujo correos electrónicos

Anexo VI: Diagrama de estados campañas

A continuación se detallan los estados por los que pasa una campaña desde su creación, a través de un diagrama de estados:

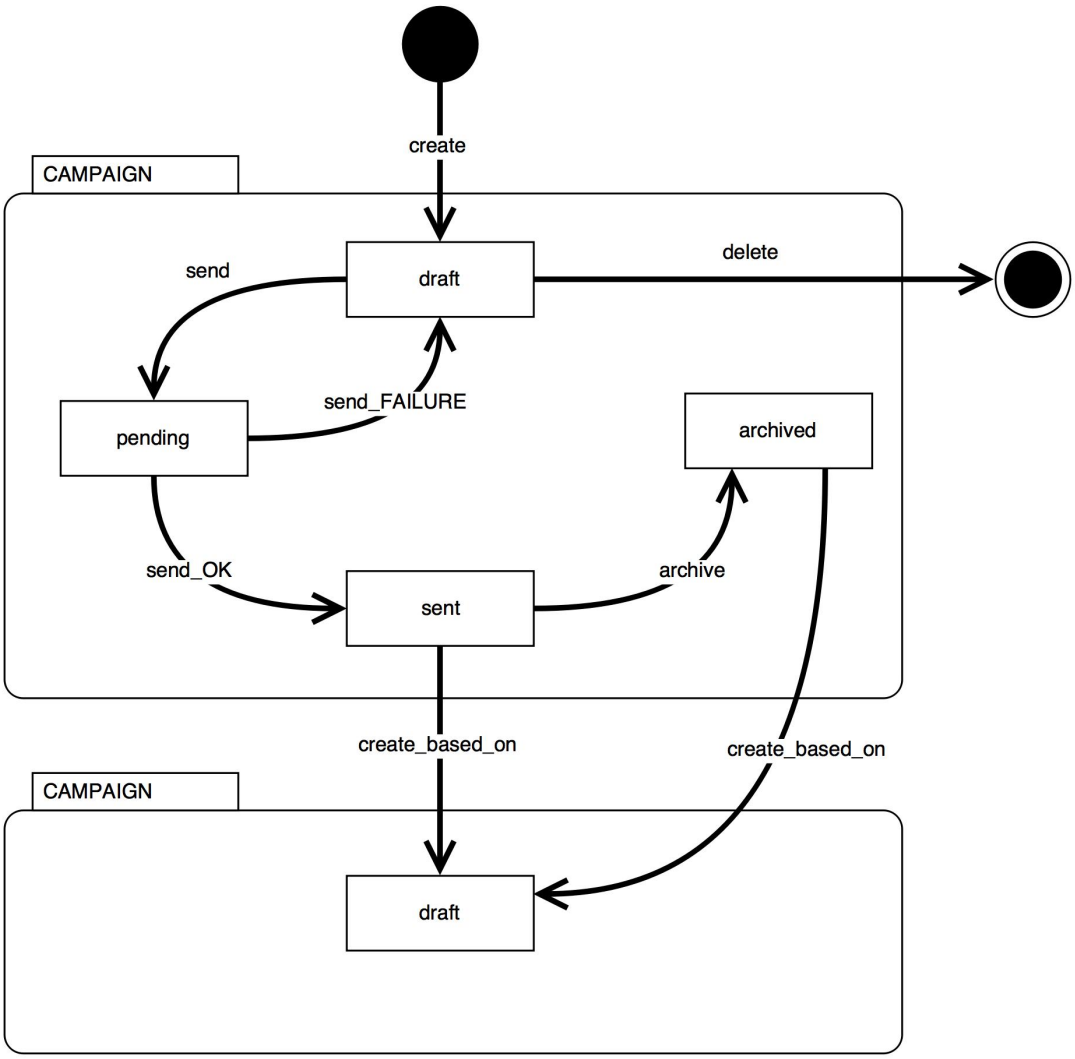


Figura 12. Diagrama estado: campañas

Anexo VII: Diagrama de estados correos electrónicos

A continuación se detallan los estados por los que pasa un correo electrónico desde su envío, a través de un diagrama de estados:

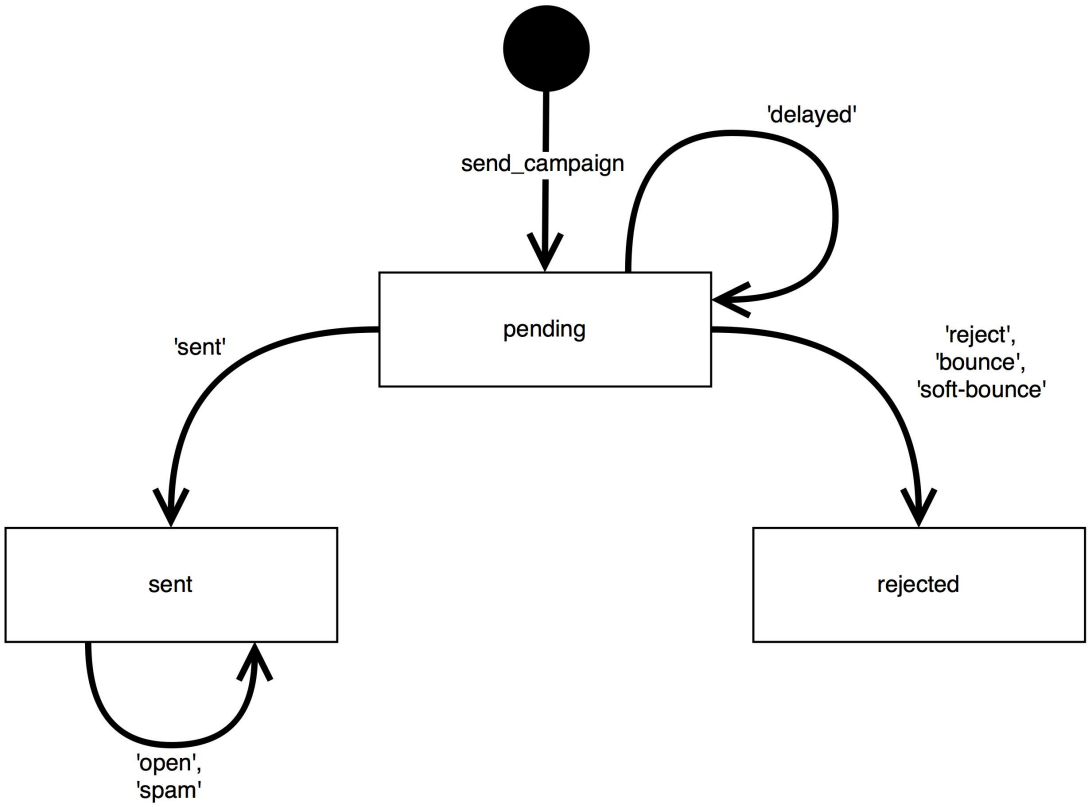


Figura 13. Diagrama estados: emails

Anexo VIII: Documentación del API Web

Anexo VIII.I: API Externa – Plantillas

Crear plantilla

Crea una plantilla.

POST `/api/ext/v0.1/template`

request – http body

Campo	Tipo	Descripción
`name`	String	Nombre de la plantilla.
`description` *opcional*	String	Descripción de la plantilla.
`htmlCode`	String	Contenido HTML de la plantilla.
`contentBlocks` *opcional*	Object[]	Lista que contiene información de los bloques de contenido editables de una plantilla.
`htmlName`	String	Identificador del bloque de contenido editable dentro del contenido HTML de la plantilla, `htmlCode`.
`name`	String	Nombre del bloque de contenido. No tiene que ver con el identificador de la plantilla, es simplemente de carácter informativo.
`description` *opcional*	String	Descripción del bloque de contenido. Cualquier tipo de información adicional asociada al bloque.

request – http body (ejemplo)

```
{
  "name": "nombre plantilla",
  "description": "descripción plantilla",
  "htmlCode": "<h1>Hola!</h1><div mc:edit=\"cblock1\"></div>",
  "contentBlocks": [
    {
      "htmlName": "cblock1",
      "name": "bloque 1",
      "description": "bloque de contenido editable número 1"
    }
  ]
```

```
}
```

response – Success 200

Nombre	Descripción
`OK`	Plantilla creada correctamente.

response – Error 4xx

Nombre	Descripción
`DUPLICATED`	Ya existe una plantilla con el mismo nombre `name`.
`INVALID_DATA`	El objeto enviado en el `http body` no es válido.

Editar plantilla

Modifica los datos de una pantilla.

PUT **`/api/ext/v0.1/template/:id`**

request – url params

Campo	Tipo	Descripción
`:id`	ObjectId	Identificador de la plantilla.

request – http body

Campo	Tipo	Descripción
`_id`	ObjectId	Identificador de la plantilla.
`name` *opcional*	String	Nombre de la plantilla.
`description` *opcional*	String	Descripción de la plantilla.
`htmlCode` *opcional*	String	Contenido HTML de la plantilla.
`contentBlocks` *opcional*	Object[]	Lista que contiene información de los bloques de contenido editables de una plantilla.
`htmlName`	String	Identificador del bloque de contenido editable dentro del contenido HTML de la plantilla, `htmlCode`.

`name`	String	Nombre del bloque de contenido. No tiene que ver con el identificador de la plantilla, es simplemente de carácter informativo.
`description` *opcional*	String	Descripción del bloque de contenido. Cualquier tipo de información adicional asociada al bloque.

request – http body (ejemplo)

```
{
  "_id": "123456789012345678901234",
  "htmlCode": "<h1>Título</h1><div mc:edit=\"cblock1\"></div>",
  "contentBlocks": [
    {
      "htmlName": "cblock1",
      "name": "bloque 1",
      "description": "bloque de contenido editable número 1"
    }
  ]
}
```

response – Success 200

Nombre	Descripción
`OK`	Plantilla modificada correctamente.

response – Error 4xx

Nombre	Descripción
`NOT_FOUND`	La plantilla con identificador `:id` no existe.
`DUPLICATED`	Ya existe una plantilla con el mismo nombre `name`.
`INVALID_DATA`	El objeto enviado en el `http body` no es válido.

Eliminar plantilla

Elimina una plantilla

DELETE `/api/ext/v0.1/template/:id`

request – url params

Campo	Tipo	Descripción
`:id`	ObjectId	Identificador de la plantilla.

response – Success 200

Nombre	Descripción
`OK`	Plantilla modificada correctamente.

response – Error 4xx

Nombre	Descripción
`NOT_FOUND`	La plantilla con identificador `:id` no existe.

Listar plantillas

Devuelve el listado de plantillas. La respuesta incluye toda la información de cada plantilla.

GET `/api/ext/v0.1/template`

response – Success 200 – Array[]

Campo	Tipo	Descripción
`_id`	ObjectId	Identificador de la plantilla
`name` *opcional*	String	Nombre de la plantilla.
`description` *opcional*	String	Descripción de la plantilla.
`htmlCode` *opcional*	String	Contenido HTML de la plantilla.
`metadata`	Object	Objeto que contiene información adicional de la plantilla.

createdDateTime	Date	Fecha y hora creación de plantilla.
updatedDateTime *opcional*	Date	Fecha y hora última actualización plantilla.
contentBlocks *opcional*	Object[]	Lista que contiene información de los bloques de contenido editables de una plantilla.
_id		Identificador del bloque de contenido
htmlName	String	Identificador del bloque de contenido editable dentro del contenido HTML de la plantilla, `htmlCode`.
name	String	Nombre del bloque de contenido. No tiene que ver con el identificador de la plantilla, es simplemente de carácter informativo.
description *opcional*	String	Descripción del bloque de contenido. Cualquier tipo de información adicional asociada al bloque.

response – Success 200(ejemplo)


```
[
  {
    "_id": "123456789012345678901234",
    "name": "Nombre plantilla",
    "description": "descripción plantilla",
    "htmlCode": "<h1>Título</h1><div mc:edit=\"cblock1\"></div>",
    "metadata": {
      "createdDateTime": "2015-01-24T11:45:14.054Z"
    },
    "contentBlocks": [
      {
        "_id": "a560123456ffff31fffff514"
        "htmlName": "cblock1",
        "name": "bloque 1",
        "description": "bloque de contenido editable número 1"
      }
    ]
  },
  {
    "_id": "098765432109876543210987",
```

```
    "name": "Nombre plantilla 2",
    "description": "descripción plantilla 2",
    "htmlCode": "<h1>Título</h1><div mc:edit=\"cblock\"></div>",
    "metadata": {
      "createdDateTime": "2015-01-24T11:45:14.054Z"
    },
    "contentBlocks": [
      {
        "_id": "a560123456ffff31fffff514"
        "htmlName": "cblock",
        "name": "nombre del bloque"
      }
    ]
  }
]
```

Obtener plantilla

Devuelve una plantilla. La respuesta incluye toda la información de dicha plantilla.

GET **/api/ext/v0.1/template/:id**

request – url params

Campo	Tipo	Descripción
:id	ObjectId	Identificador de la plantilla

response – Success 200

Campo	Tipo	Descripción
_id	ObjectId	Identificador de la plantilla
name *opcional*	String	Nombre de la plantilla.
description *opcional*	String	Descripción de la plantilla.
htmlCode *opcional*	String	Contenido HTML de la plantilla.
metadata	Object	Objeto que contiene información adicional de la plantilla.

createdDateTime	Date	Fecha y hora creación de plantilla.
updatedDateTime *opcional*	Date	Fecha y hora última actualización plantilla.
contentBlocks *opcional*	Object[]	Lista que contiene información de los bloques de contenido editables de una plantilla.
_id	ObjectId	Identificador del bloque de contenido
htmlName	String	Identificador del bloque de contenido editable dentro del contenido HTML de la plantilla, `htmlCode`.
name	String	Nombre del bloque de contenido. No tiene que ver con el identificador de la plantilla, es simplemente de carácter informativo.
description *opcional*	String	Descripción del bloque de contenido. Cualquier tipo de información adicional asociada al bloque.

response – Success 200(ejemplo)


```
{
  "_id": "123456789012345678901234",
  "name": "Nombre plantilla",
  "description": "descripción plantilla",
  "htmlCode": ""<h1 mc:edit=\"title"></h1><div mc:edit=\"block"></div>",
  "metadata": {
    "createdDateTime": "2015-01-24T11:45:14.054Z",
    "updatedDateTime": "2015-02-01T23:09:44.400Z"
  },
  "contentBlocks": [
    {
      "_id": "a560123456ffff3123a1a340"
      "htmlName": "title",
      "name": "título",
      "description": "esto es el título editable"
    },
    {
      "_id": "a560123456ffff31fffff514"
      "htmlName": "block",
      "name": "bloque 1",
```

```
      "description": "esto es bloque de contenido editable"
    }
  ]
}
```

response – Error 4xx

Nombre	Descripción
`NOT_FOUND`	La plantilla con identificador `:id` no existe.

Anexo VIII.II: API Externa – Campañas

Archivar campaña

Archiva una campaña previamente enviada.

PATCH **`/api/ext/v0.1/campaign/:id`**

request – url params

Campo	Tipo	Descripción
`:id`	ObjectId	Identificador de la campaña.

request – query params

Campo	Tipo	Descripción
`operation`	String	Valores permitidos: `"archive"`.

response – Success 200

Nombre	Descripción
`OK`	Campaña archivada correctamente.

response – Error 4xx

Nombre	Descripción
`NOT_FOUND`	La campaña con identificador `:id` no existe.
`BAD_REQUEST`	La petición realizada al método es incorrecta.

Crear campaña

Crea una campaña.

POST **`/api/ext/v0.1/campaign`**

request – http body

Campo	Tipo	Descripción
`name`	String	Nombre de la campaña.
`description` *opcional*	String	Descripción de la campaña.

`subject` *opcional*	String	Asunto con el que serán enviados los correos electrónicos de la campaña.
`fromEmail`	String	Dirección de correo electrónico que aparecerá como remitente en los correos electrónicos de la campaña que serán enviados.
`fromName` *opcional*	String	Nombre que aparecerá como remitente en los correos electrónicos de la campaña que serán enviados.
`segmentation`	Number[]	Lista que contiene los identificadores de segmentación a los que va dirigida la campaña.
`template`	Object	Objeto que contiene información sobre qué plantilla se usará en la campaña y su contenido.
`_id`	ObjectId	El identificador de la plantilla que va a utilizar la campaña.
`contentBlocks` *opcional*	Object[]	Lista que contiene el contenido de cada bloque de contenido editable de la plantilla para la campaña.
`_id`	ObjectId	Identificador de bloque de contenido editable.
`content`	String	El contenido que será insertado en el bloque de contenido editable especificado mediante el campo `_id`.

request – http body (ejemplo)

```
{
  "name": "Nombre campaña",
  "description": "descripción campaña",
  "subject": "asunto de los correos electrónicos",
  "fromEmail": "example@example.com",
  "fromName": "John Doe",
  "segmentation": [1234, 55, 98],
  "template": {
    "_id": "098765432109876543210987",
    "contentBlocks": [
      {
        "_id": "a560123456ffff3123a1a340",
        "content": "este será el contenido de a560123456ffff3123a1a340"
      },
```

```
      {
        "_id": "a560123456ffff31fffff514",
        "content": "este será el contenido de a560123456ffff31fffff514"
      }
    ]
  }
}
```

response – Success 200

Nombre	Descripción
`OK`	Campaña creada correctamente.

response – Error 4xx

Nombre	Descripción
`DUPLICATED`	Ya existe una campaña con el mismo nombre `name`.
`INVALID_DATA`	El objeto enviado en el `body` no es válido.

Crear campaña basada

Crea una campaña basada en otra campaña existente.

`GET /api/ext/v0.1/campaign/actions`

request – query params

Campo	Tipo	Descripción
`id`	ObjectId	Identificador de la campaña existente.
`type`	String	Valores permitidos: `"createBasedOn"`.

response – Success 200

Nombre	Descripción
`OK`	Campaña a partir de campaña existente creada correctamente.

response – Error 4xx

Nombre	Descripción
NOT_FOUND	La campaña con identificador :id no existe.
BAD_REQUEST	La petición realizada al método es incorrecta.

Editar campaña

Modifica los datos de una campaña.

PUT **/api/ext/v0.1/campaign/:id**

request – url params

Campo	Tipo	Descripción
:id	ObjectId	Identificador de la campaña.

request – http body

Campo	Tipo	Descripción
name *opcional*	String	Nombre de la campaña.
description *opcional*	String	Descripción de la campaña.
subject *opcional*	String	Asunto con el que serán enviados los correos electrónicos de la campaña.
fromEmail *opcional*	String	Dirección de correo electrónico que aparecerá como remitente en los correos electrónicos de la campaña que serán enviados.
fromName *opcional*	String	Nombre que aparecerá como remitente en los correos electrónicos de la campaña que serán enviados.
segmentation *opcional*	Number[]	Lista que contiene los identificadores de segmentación a los que va dirigida la campaña.
template *opcional*	Object	Objeto que contiene información sobre qué plantilla se usará en la campaña y su contenido.
contentBlocks *opcional*	Object[]	Lista que contiene el contenido de cada bloque de contenido editable de la plantilla para la campaña.

`_id`	ObjectId	Identificador de bloque de contenido editable.
`content`	String	El contenido que será insertado en el bloque de contenido editable especificado mediante el campo `_id`.

request – http body (ejemplo)

```
{
  "name": "Nombre campaña modificado",
  "fromEmail": "foobar@example.com",
  "fromName": "Foo Bar",
  "template": {
    "contentBlocks": [
      {
        "_id": "a560123456ffff3123a1a340",
        "content": "este será el nuevo contenido de a560123456ffff3123a1a340"
      }
    ]
  }
}
```

response – Success 200

Nombre	Descripción
`OK`	Campaña modificada correctamente.

response – Error 4xx

Nombre	Descripción
`NOT_FOUND`	No se ha llevado a cabo ninguna modificación en la campaña solicitada debido a algún motivo como que no existe ninguna campaña con identificador `:id` o el estado `status` actual de la campaña no lo permite.

Eliminar campaña

Elimina una campaña.

`DELETE` `/api/ext/v0.1/campaign/:id`

request – url params

Campo	Tipo	Descripción
`:id`	ObjectId	Identificador de la campaña.

response – Success 200

Nombre	Descripción
`OK`	Campaña eliminada correctamente.

response – Error 4xx

Nombre	Descripción
`NOT_FOUND`	La campaña con identificador `:id` no existe.

Lanzar campaña

Realiza el lanzamiento de una campaña.

`GET /api/ext/v0.1/campaign/actions`

request – query params

Campo	Tipo	Descripción
`id`	ObjectId	Identificador de la campaña que desea lanzarse.
`type`	String	Valores permitidos: `"sendCampaign"`.

response – Success 200

Nombre	Descripción
`OK`	Campaña lanzada correctamente.

response – Error 4xx

Nombre	Descripción
`NOT_FOUND`	La campaña con identificador `:id` no existe.
`BAD_REQUEST`	La petición realizada al método es incorrecta.

Listar campañas

Devuelve el listado de campañas. La respuesta incluye toda la información de cada campaña.

`GET` **`/api/ext/v0.1/campaign`**

request – query params

Campo	Tipo	Descripción
`filter` *opcional*	String	Filtra los elementos del listado a nivel de estado de una campaña (la propiedad `status` de una campaña). Valores permitidos: `"draft"`, `"sent"` y `"archived"`.
`orderBy` *opcional*	String	Campo mediante el cual se ordenará el listado.
`pageSize` *opcional*	String	Tamaño de cada bloque de paginación.
`page` *opcional*	String	Bloque de paginación que queremos obtener.

response – Success 200 – Array[]

Campo	Tipo	Descripción
`Array0`	Object[]	`response.body[0]` contiene el listado de campañas.
`_id`	ObjectId	Identificador de la campaña.
`name`	String	Nombre de la campaña.
`description`	String	Descripción de la campaña.
`status`	String	Estado en el que se encuentra la campaña.
`createdDateTime`	Date	Fecha y hora creación de la campaña.
`subject` *opcional*	String	Asunto con el que serán enviados los correos electrónicos de la campaña.
`fromEmail`	String	Dirección de correo electrónico que aparecerá como remitente en los correos electrónicos de

		la campaña que serán enviados.
`fromName` *opcional*	String	Nombre que aparecerá como remitente en los correos electrónicos de la campaña que serán enviados.
`segmentation`	Number[]	Lista que contiene los identificadores de segmentación a los que va dirigida la campaña.
`template`	Object	Plantilla de correo electrónico de la campaña.
`name`	String	Nombre de la plantilla.
`description` *opcional*	String	Descripción de la plantilla.
`htmlCode`	String	Contenido HTML de la plantilla.
`metadata`	Object	Objeto que contiene información adicional de la plantilla.
`createdDateTime`	Date	Fecha y hora creación plantilla.
`updatedDateTime` *opcional*	Date	Fecha y hora última actualización de plantilla.
`contentBlocks` *opcional*	Object[]	Lista que contiene información de los bloques de contenido editables de una plantilla.
`_id`	ObjectId	Identificador de bloque de contenido editable.
`htmlName`	String	Identificador del bloque de contenido editable dentro del contenido HTML de la plantilla, `htmlCode`.
`name`	String	Nombre del bloque de contenido. No tiene que ver con el identificador de la plantilla, es simplemente de carácter informativo.
`description` *opcional*	String	Descripción del bloque de contenido. Cualquier tipo de información adicional asociada al bloque, también de carácter informativo.
`content` *opcional*	String	Contenido que se insertará en el bloque al enviarse la campaña.
`Array1`	Object[]	`response.body[1]` contiene información relacionada con el listado de campañas.
`count`	Number	Número de campañas que contiene el listado,

		response.body[0]

response – Success 200(ejemplo)


```
[
  [
    {
      "_id": "123456789012345678901234",
      "name": "Nombre campaña",
      "description": "Descripción campaña",
      "status": "draft",
      "createdDateTime": "2015-01-24T11:45:14.054Z",
      "subject": "asunto de los correos electrónicos",
      "fromEmail": "example@example.com",
      "fromName": "John Doe",
      "segmentation":[1234, 99],
      "template": {
        "name": "Nombre plantilla",
        "description": "descripción plantilla",
        "htmlCode": "<h1 mc:edit=\"block1\"></h1><div
mc:edit=\"block2\"></div>",
        "metadata": {
          "createdDateTime": "2015-01-24T11:58:56.111Z",
          "updatedDateTime": "2015-01-25T23:04:14.166Z",
        },
        "contentBlocks": [
          {
            "_id": "11111abf0023674cccc11111",
            "htmlName": "block1",
            "name": "bloque de contenido 1",
            "description": "bloque de contenido editable 1",
            "content": "esto irá en el bloque 1"
          },
          {
            "_id": "11111abf0023674cccc11111",
            "htmlName": "block2",
            "name": "Bloque de contenido 2",
            "description": "bloque de contenido editable 2",
            "content": "esto irá en el bloque 2"
          }
```

```
        ]
      }
    },
    {
      "_id": "123456789012345678904321",
      "name": "Nombre campaña",
      "description": "Descripción campaña",
      "status": "draft",
      "createdDateTime": "2015-01-25T23:04:00.000Z",
      "subject": "asunto de los correos electrónicos",
      "fromEmail": "example@example.com",
      "fromName": "John Doe",
      "segmentation":[12],
      "template": {
        "name": "Nombre plantilla",
        "description": "descripción plantilla",
        "htmlCode": "<div mc:edit=\"block\"></div>",
        "metadata": {
          "createdDateTime": "2015-02-01T11:23:33.134Z",
        },
        "contentBlocks": [
          {
            "_id": "11111abf0023674cccc11111",
            "htmlName": "block",
            "name": "bloque de contenido",
            "description": "bloque de contenido editable",
            "content": "esto irá en el bloque"
          }
        ]
      }
    }
  ],
  [
    {
      count: 2
    }
  ]
]
```

Obtener campaña

Devuelve una campaña. La respuesta incluye toda la información de dicha campaña.

GET `/api/ext/v0.1/campaign/:id`

request – url params

Campo	Tipo	Descripción
`:id`	ObjectId	Identificador de la campaña.

response – Success 200

Campo	Tipo	Descripción
`_id`	ObjectId	Identificador de la campaña.
`name`	String	Nombre de la campaña.
`description`	String	Descripción de la campaña.
`status`	String	Estado en el que se encuentra la campaña.
`createdDateTime`	Date	Fecha y hora creación de la campaña.
`subject` *opcional*	String	Asunto con el que serán enviados los correos electrónicos de la campaña.
`fromEmail`	String	Dirección de correo electrónico que aparecerá como remitente en los correos electrónicos de la campaña que serán enviados.
`fromName` *opcional*	String	Nombre que aparecerá como remitente en los correos electrónicos de la campaña que serán enviados.
`segmentation`	Number[]	Lista que contiene los identificadores de segmentación a los que va dirigida la campaña.
`template`	Object	Plantilla de correo electrónico de la campaña.
`name`	String	Nombre de la plantilla.
`description` *opcional*	String	Descripción de la plantilla.
`htmlCode`	String	Contenido HTML de la plantilla.

metadata	Object	Objeto que contiene información adicional de la plantilla.
createdDateTime	Date	Fecha y hora creación plantilla.
updatedDateTime *opcional*	Date	Fecha y hora última actualización de plantilla.
contentBlocks *opcional*	Object[]	Lista que contiene información de los bloques de contenido editables de una plantilla.
_id	ObjectId	Identificador de bloque de contenido editable.
htmlName	String	Identificador del bloque de contenido editable dentro del contenido HTML de la plantilla, `htmlCode`.
name	String	Nombre del bloque de contenido. No tiene que ver con el identificador de la plantilla, es simplemente de carácter informativo.
description *opcional*	String	Descripción del bloque de contenido. Cualquier tipo de información adicional asociada al bloque, también de carácter informativo.
content *opcional*	String	Contenido que se insertará en el bloque al enviarse la campaña.

response – Success 200(ejemplo)


```
{
  "_id": "123456789012345678901234",
  "name": "Nombre campaña",
  "description": "Descripción campaña",
  "status": "draft",
  "createdDateTime": "2015-01-24T11:45:14.054Z",
  "subject": "asunto de los correos electrónicos",
  "fromEmail": "example@example.com",
  "fromName": "John Doe",
  "segmentation":[1234, 99],
  "template": {
    "name": "Nombre plantilla",
    "description": "descripción plantilla",
    "htmlCode": "<h1 mc:edit=\"block1\"></h1><div mc:edit=\"block2\"></div>",
    "metadata": {
```

```
      "createdDateTime": "2015-01-24T11:58:56.111Z",
      "updatedDateTime": "2015-01-25T23:04:14.166Z",
    },
    "contentBlocks": [
      {
        "_id": "11111abf0023674cccc11111",
        "htmlName": "block1",
        "name": "bloque de contenido 1",
        "description": "bloque de contenido editable 1",
        "content": "esto irá en el bloque 1"
      },
      {
        "_id": "11111abf0023674cccc11111",
        "htmlName": "block2",
        "name": "Bloque de contenido 2",
        "description": "bloque de contenido editable 2",
        "content": "esto irá en el bloque 2"
      }
    ]
  }
}
```


response – Error 4xx

Nombre	Descripción
NOT_FOUND	La campaña con identificador `:id` no existe.

Anexo VIII.III: API Externa – Analíticas

Obtener analíticas

Devuelve las analíticas básicas de una campaña.

`GET /api/ext/v0.1/campaign/:id/stats`

request – url params

Campo	Tipo	Descripción
`:id`	ObjectId	Identificador de la campaña.

response – Success 200

Campo	Tipo	Descripción
`total`	Number	Número total de correos electrónicos.
`pending`	Number	Número de correos pendientes de envío.
`sent`	Number	Número de correos enviados.
`rejected`	Number	Número de correos rechazados y no enviados.
`opened`	Number	Número de correos que han sido abiertos.
`marked_as_spam`	Number	Número de correos que han sido marcados como *spam*.

Anexo VIII.IV: API Requerida

Esta API la deberán implementar los clientes de *Baboon* para poder realizar envíos de campañas. Consta de un único método, y deberán de impementarlo todas las aplicaciones dadas de alta en el sistema que vayan a utilizar los servicios que la plataforma ofrece. Este método se detalla a continuación:

Carga de destinatarios

Devuelve el listado de destinatarios a los que va dirigida la campaña.

*Deberá implementarse un método de tipo **GET** que cumpla la siguiente especificación:*

request – url params

Campo	Tipo	Descripción
`segmentation`	Number[]	Lista que contiene los identificadores de segmentación a los que va dirigida la campaña. *Los destinatarios que devuelva el método deben pertenecer a alguno de los segmentos especificados.*

request – url (example)

```
GET        /endpoint/method?segmentation=[1234,55]
```

La respuesta al método, deberá de ir en formato *JSON*. A continuación se explica detalladamente qué estructura deberá de tener y se proporcionará un ejemplo de respuesta válida:

response – Success 200 – Array[]

Campo	Tipo	Descripción
`segmentation`	Number	Identificador de segmentación al que pertenece el destinatario.
`email`	String	Dirección de correo del destinatario.
`metadata`	Object[]	Listado de propiedades asociadas al destinatario.
`name`	String	Nombre de propiedad.
`content`	String	El valor de la propiedad de nombre `name`.

response – Success 200(ejemplo)


```
[
  {
    "segmentation": 1234,
    "email": "johndoe@example.com",
    "metadata": [
      {
        "name": "name",
        "content": "John"
      },
      {
        "name": "Surname",
        "content": "Doe"
      }
    ],
  },
  {
    "segmentation": 1234,
    "email": "foobar@example.com",
    "metadata": [
      {
        "name": "name",
        "content": "Foo"
      },
      {
        "name": "Surname",
        "content": "Bar"
      }
    ],
  },
  {
    "segmentation": 55,
    "email": "travis@example.com",
    "metadata": [
      {
        "name": "name",
        "content": "Travis"
      }
    ]
```

```
  }
]
```

Anexo VIII.V: API Expuesta

Esta API se ha implementado para poder registrar los eventos asociados a los correos electrónicos que el servicio externo *Mandrill* nos envía. Consta de un único método, y se detalla a continuación:

Registro de eventos

Registra los evento asociado a un correo electrónico enviado en una campaña.

POST `/api/webhook/event`

Mandrill envía en el `body` de la petición un objeto en formato *JSON* con información asociada al evento. El formato que recibe este método se detalla a continuación:

request – http body

Campo	Tipo	Descripción
`ts`	Number	Timestamp en formato Unix UTC de cuando se produjo el evento.
`event`	String	El tipo de evento que se ha producido: enviado, abierto, marcado como *spam*....
`_id`	String	El identificador único del mensaje que generó el evento.
`msg`	Object	Detalles acerca del mensaje que produjo el evento: destinatario, timestamp de envío, asunto del correo, histórico de eventos producidos anteriormente...

Tras realizar el registro del evento y atender la petición, el método devuelve el código de respuesta **200 – OK**.

Anexo IX: EditorConfig

EditorConfig ayuda a los desarrolladores a definir y mantener estilos de codificación consistentes entre diferentes editores e IDEs. El proyecto *EditorConfig* consta de un formato de fichero que permite definir estilos de codificación y de una colección de plugins de editores de texto que les posibilita la lectura e interpretación de dicho fomato de fichero y su cumplimiento dentro de un mismo proyecto. Los ficheros de *EditorConfig* son sencillos de leer y se llevan bien con los sistemas de versiones.

Cuando un editor de texto abre un archivo, los plugins *EditorConfig* buscan un fichero de nombre **`.editorconfig`** en dicha ruta y en los directorios padres. La búsqueda de este fichero termina si se alcanza el directorio raíz del proyecto o en caso de que se haya encontrado dicho fichero que contenga la propiedad **`root=true`**.

Estos ficheros son leídos de arriba abajo y los ficheros *EditorConfig* más próximos a la ruta del fichero abierto por el editor de textos son leídos en último lugar. Las propiedades de los distintos ficheros **`.editorconfig`** encontrados se aplican en el orden en el que son leídas; por lo tanto, las propiedades de los ficheros más próximos tienen preferencia sobre las demás.

.editorconfig

Dentro del proyecto se ha definido el siguiente fichero sencillo de *EditorConfig*:

```
# top-most EditorConfig file
root = true

# matches any string of characters, except path separators (/)
[*]

indent_style = space
indent_size = 2

end_of_line = lf
charset = utf-8
trim_trailing_whitespace = true
insert_final_newline = true

# matches any string of characters, except path separators (/),
# ending in ‘.md’
[*.md]

trim_trailing_whitespace = false
```

Propiedades del fichero definido:

- **indent_style:** tipo de indentación (tabulador o espacios)
- **indent_size:** número de columnas utilizadas para cada nivel de indentación (número de tabuladores o espacios)
- **end_of_line:** representación del fin de línea
- **charset:** tipo de set de caracteres
- **trim_trailing_whitespace:** eliminación de caracteres de espacios que preceden a un salto de línea
- **insert_final_newline:** forzado a que el fichero acabe con una línea en blanco

Anexo X: JSON Web Tokens

Funcionamiento de los JSON Web Tokens

Cuando un usuario se loguea utilizando sus credenciales, se devolverá un *JSON Web Token* que deberá de guardar localmente, en vez de la aproximación tradicional de crear una sesión en el servidor.

Siempre que el usuario quiera acceder a una ruta o recurso protegido, deberá de enviar el *JWT*, típicamente en la cabecera *Authorization* utilizando el *Bearer schema*. El contenido de la cabecera, deberá de aparentar de la siguiente manera:

```
Authorization: Bearer <token>
```

Este es un mecanismo de autenticación sin estado, debido a que el estado del usuario no es almacenado en la memoria del servidor. Las rutas del servidor protegidas deberán de comprobar la presencia del token y, en caso de estar presente y ser válido, dar acceso al recurso solicitado.

El diagrama presentado a continuación, muestra este proceso:

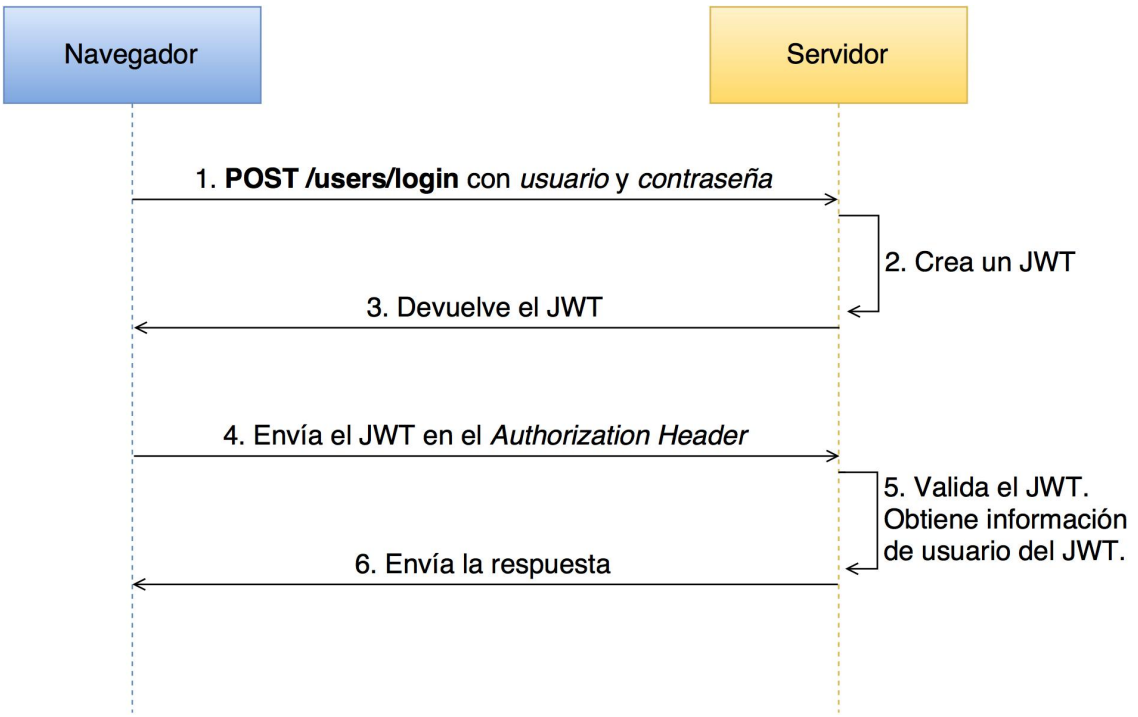


Figura 14. Funcionamiento básico JWT

Como los tokens son almacenados en el lado del cliente, no hay información de estado y la aplicación se vuelve totalmente escalable. Por supuesto, podemos hacer que el token expire después de un tiempo, lo que añade una capa extra de seguridad.

Estructura de un JWT

Un *JWT* consiste de 3 partes, separadas por un punto '**.**' que son:

- **Header**
- **Payload**
- **Signature**

Un *JWT*, por lo tanto, tendría el siguiente aspecto:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI1NGE4Y2U2MThlOTFiMGIxMzY2NWUyZjkiLCJpYXQiOiIxNDI0MTgwNDg0IiwiZXhwIjoiMTQyNTM5MDE0MiJ9.yk4nouUteW54F1HbWtgg1wJxeDjqDA_8AhUPyjE5K0U
```

Header

El *header* típicamente consiste de dos partes: el tipo de token, que en este caso será *JWT*, y el algoritmo de codificación empleado que comúnmente se trata del *HMAC SHA256*. Por ejemplo:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

A continuación, este *JSON* es codificado en **Base64** para pasar a formar la primera parte del *JWT*.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
```

Payload

La segunda parte del token es el *payload*, que contiene los llamados *claims*, una serie de atributos que definen el token y que son de carácter opcional pero su especificación es recomendable. Un ejemplo de *payload* sería:

```
{
  "sub": "54a8ce618e91b0b13665e2f9",
  "iat": "1424180484",
  "exp": "1425390142"
}
```

Este *JSON* es codificado en **Base64** y pasa a formar la segunda parte del *JWT*.

```
eyJzdWIiOiI1NGE4Y2U2MThlOTFiMGIxMzY2NWUyZjkiLCJpYXQiOiIxNDI0MTgwNDg0IiwiZXhwIjoiMTQyNTM5MDE0MiJ9
```

Signature

La firma es la tercera y última parte del *JSON Web Token*, está formada por los anteriores componentes (*header* y *payload*) cifrados en **Base64** con una clave secreta (almacenada en nuestro backend):

```
HMACSHA256(
    base64UrlEncode(header) + "." +
    base64UrlEncode(payload), secret
);
```

De esta forma, esta tercera parte del *JWT* es utilizada para verificar que el remitente del token es quien dice ser y asegurarnos de que el mensaje no ha sido modificado durante de camino.

yk4nouUteW54F1HbWtgg1wJxeDjqDA_8AhUPyjE5K0U

Anexo XI: Tests

En este anexo se detallan los tests que han sido implementados para cada funcionalidad del sistema. Se muestra concretamente la descripción de todos los tests directamente del código Javascript con **Mocha**, suprimiendo el contenido de las funciones *callback*.

Gestión de usuarios

En esta sección se especifican los tests relacionados con la gestión de usuarios administradores del sistema:

```
describe('CREATE AN ADMIN', function() {


  it('should be able to create an admin', function(done) {
  });


  it('should not be able to create an admin with an already existing name',
function(done) {
  });


  it('should not be able to create an admin when passing junk',
function(done) {
  });


});


describe('GET A SINGLE ADMIN', function() {


  it('should be getting an existing admin', function(done) {
  });


  it('should be getting nothing when looking up for non-existing admin',
function(done) {
  });


});


describe('GET ADMIN LIST', function() {


  it('should be getting a list of ALL admins', function(done) {
  });


});
```

```
describe('DELETE AN ADMIN', function() {

  it('should be deleting an existing admin', function(done) {
  });

  it('should be doing nothing when wanting to delete a non-existing admin',
function(done) {
  });

});

describe('UPDATE AN EXISTING ADMIN', function() {

  it('should be updating an existing admin', function(done) {
  });

  it('should NOT be updating an existing admin if passed junk',
function(done) {
  });

  it('should be doing nothing when wanting to update a non-existing admin',
function(done) {
  });

});

describe('UPDATE ADMIN PASSWORD', function() {

  it('should be updating an admin\'s pwd', function(done) {
  });

  it('should be doing nothing when wanting to update a non-existing admin\'s
pwd', function(done) {
  });

});

describe('RETURN ADMIN INFO', function() {

  it('should return the admin\'s info', function(done) {
```

```
  });

});

describe('CHECKING IF A GIVEN ADMIN EMAIL IS VALID', function() {

  it('checkValidEmail: should return TRUE when email sent is VALID',
function(done) {
  });

  it('checkValidEmail: should return FALSE when email sent is INVALID',
function(done) {
  });

});

describe('PASSWORD FORGOTTEN', function() {

  it('forgotPwd: should return OK when password forgotten', function(done) {
  });

});

describe('RESET PASSWORD', function() {

  it('resetPwd: should return OK when resetting password', function(done) {
  });

  it('resetPwd: should not reset when passing wrong "forgotPwdToken"',
function(done) {
  });

  it('resetPwd: should not reset when passing expired "forgotPwdToken"',
function(done) {
  });

});
```

Gestión de aplicaciones

En esta sección se especifican los tests relacionados con la gestión de aplicaciones:

```
describe('CREATE AN APPLICATION', function() {


  it('should be able to register an app', function(done) {
  });


  it('should not be able to register an app with an already existing name',
function(done) {
  });


  it('should not be able to register an app when passing junk',
function(done) {
  });


});


describe('GET A SINGLE APPLICATION', function() {


  it('should be getting an existing app', function(done) {
  });


  it('should be getting nothing when looking up for non-existing app',
function(done) {
  });


});


describe('GET APPLICATION LIST', function() {


  it('[NO filter] should be getting a list of ALL apps (enabled & disabled)',
function(done) {
  });


  it('[filter="disabled"] should be getting a list only of the DISABLED
apps', function(done) {
  });


  it('[filter="enabled"] should be getting a list only of the ENABLED apps',
function(done) {
  });
```

```
});

describe('DISABLE AN APPLICATION', function() {

  it('should be disabling an existing app', function(done) {
  });

  it('should be doing nothing when wanting to disable a non-existing app',
function(done) {
  });

});

describe('UPDATE AN EXISTING APPLICATION', function() {

  it('should be updating an existing app', function(done) {
  });

  it('should NOT be updating an existing app if passed junk', function(done)
{
  });

  it('should be doing nothing when wanting to update a non-existing app',
function(done) {
  });

});

describe('ENABLING A DISABLED APPLICATION', function() {

  it('should be enabling a disabled app', function(done) {
  });

  it('should be doing nothing when wanting to enable a non-existing app',
function(done) {
  });

});

describe('REGENERATING APPLICATION KEYS', function() {
```

```
  it('should be regenerating the keys of an existing application',
function(done) {
  });


  it('should be doing nothing when wanting to regenerate keys of a non-
existing app', function(done) {
  });


});


describe('CHECKING IF A GIVEN APP NAME IS VALID', function() {


  it('checkValidName [No ID param]: should return TRUE when name sent is
VALID', function(done) {
  });


  it('checkValidName [No ID param]: should return FALSE when name sent is NOT
VALID', function(done) {
  });


  it('checkValidName [ID param]: should return TRUE when name sent is it\'s
name', function(done) {
  });


});
```

Gestión de plantillas

En esta sección se especifican los tests relacionados con la gestión de plantillas:

```
describe('CREATE A TEMPLATE', function() {


  it('should be able to create a template', function(done) {
  });


  it('should not be able to create a template with an already existing name',
function(done) {
  });


  it('should not be able to create a template when passing junk',
function(done) {
  });


});


describe('GET A SINGLE TEMPLATE', function() {


  it('should be getting an existing template', function(done) {
  });


  it('should be getting nothing when looking up for non-existing template',
function(done) {
  });


});


describe('GET TEMPLATES LIST', function() {


  it('should be getting the list of templates of the given app',
function(done) {
  });


});


describe('DELETE AN APP\'s TEMPLATE', function() {


  it('should be deleting an existing app\'s template', function(done) {
  });
```

```
  it('should be doing nothing when wanting to delete a non-existing app\'s
template', function(done) {
  });


});


describe('UPDATE AN EXISTING TEMPLATE', function() {


  it('should be updating an existing template', function(done) {
  });


  it('should NOT be updating an existing template if passed junk',
function(done) {
  });


  it('should be doing nothing when wanting to update a non-existing
template', function(done) {
  });


  it('should NOT be updating an existing template if name is duplicated',
function(done) {
  })


});
```

Gestión de campañas

En esta sección se especifican los tests relacionados con la gestión de campañas:

```
describe('CREATE A CAMPAIGN', function() {


  it('should be able to create a campaign', function(done) {
  });


  it('should not be able to create a campaign with an already existing name',
function(done) {
  });


  it('should not be able to create a campaign when passing junk',
function(done) {
  });


});


describe('GET A SINGLE CAMPAIGN', function() {


  it('should be getting an existing campaign', function(done) {
  });


  it('should be getting nothing when looking up for non-existing campaign',
function(done) {
  });


});


describe('GET CAMPAIGNS LIST', function() {


  it('[NO filter] should be getting a list of ALL campaigns', function(done)
{
  });


  it('[filter="draft"] should be getting a list only of the DRAFT campaigns',
function(done) {
  });


  it('[filter="sent"] should be getting a list only of the SENT campaigns',
function(done) {
  });
```

```
  it('[filter="archived"] should be getting a list only of the ARCHIVED campaigns', function(done) {
  });

});

describe('DELETE A DRAFT CAMPAIGN', function() {

  it('should be deleting an existing campaign', function(done) {
  });

  it('should be doing nothing when wanting to delete a non-existing campaign', function(done) {
  });

});

describe('ARCHIVING A SENT CAMPAIGN', function() {

  it('should be archiving a sent campaign', function(done) {
  });

  it('should be doing nothing when wanting to archive a non-existing campaign', function(done) {
  });

});

describe('UPDATE AN EXISTING CAMPAIGN', function() {

  it('should be updating an existing campaign', function(done) {
  });

  it('should NOT be updating an existing template if passed junk', function(done) {
  });

  it('should be doing nothing when wanting to update a non-existing campaign', function(done) {
```

```
  });

});

describe('CREATE A BASED ON CAMPAIGN', function() {

  it('should be creating a based on campaign', function(done) {
  });

  it('should be doing nothing when passing non-existing id', function(done) {
  });

});

describe('SEND CAMPAIGN', function() {

  it('should be sending a campaign', function(done) {
  });

  it('should be doing nothing when passing non-existing id', function(done) {
  });

});
```

Estadísticas de campañas

En esta sección se especifican los tests relacionados con la gestión de plantillas:

```
describe('GET CAMPAIGN STATS', function() {


  it('should be able to obtain a campaign\'s stats', function(done) {
  });


  it('should be getting nothing when looking up for non-existing campaign\'s
stats', function(done) {
  });


});
```

Anexo XII: Configuración por entornos

Los principales ficheros de configuración son:

- **/server/config/environment/index.js**
- **/server/config/environment/development.js**
- **/server/config/environment/preproduction.js**
- **/server/config/environment/production.js**
- **/server/config/environment/test.js**

/server/config/environment/index.js

```
// All configurations will extend these options
// ============================================
module.exports = {
  env: process.env.NODE_ENV,

  // Root path of server
  root: path.normalize(__dirname + '/../../..'),

  // Server port
  port: process.env.PORT || 3333,

  // Should we populate the DB with sample data?
  seedDB: false,

  // MongoDB connection options
  mongo: {
    options: {
      db: {
        safe: true
      }
    }
  }
};
```

Variables de utilidad para la configuración:

- **env:** Define el entorno de ejecución
- **root:** Directorio raíz donde se encuentra el servidor
- **port:** Puerto del servidor
- **seedDB:** Indica si debe de inicializarse la base de datos con datos de prueba definidos en **/server/config/seed.js**

- **mongo.options:** Opciones de acceso a la base de datos.

/server/config/environment/development.js

```
// Development specific configuration
// ==================================
module.exports = {
  // MongoDB connection options
  mongo: {
    uri: 'mongodb://localhost/baboon-dev'
  },

  mail: {
    host: 'smtp.gmail.com',
    port: 465,
    from: '<10labs.test@gmail.com>',
    to: '<10labs.test@gmail.com>',
    subjectForgetPwd: 'Olvido contraseña Baboon',
    secure: true,
    auth: {
      user: '10labs.test@gmail.com',
      pass: 'XXXXXXXX'
    }
  },

  mandrill: {
    apikey: 'XXXXXXXX',
    webhook: {
      key: 'XXXXXXXX',
      url: 'http://a5851c86.ngrok.io/api/webhook/event'
    }
  },

  logger: {
    transports: {
      file: {
        options: {
          filename: rootPath + '/logs/baboon.log',
          prettyPrint: true,
          json: false
        }
      }
```

```
    }
  }
};
```

Variables de utilidad para la configuración:

- **mongo.uri:** Base de datos mongodb
- **mail.host:** Servidor de correo
- **mail.port:** Puerto del servidor de correo
- **mail.from:** Puerto del servidor
- **mail.to:** Destinatario del correo electrónico
- **mail.subjectForgetPwd:** Asunto de los correos de olvido de contraseña
- **mail.secure:** Opción de conexión segura
- **mail.auth.user:** Usuario de corre
- **mail.auth.pass:** Contraseña del usuario de correo
- **mandrill.apikey:** API Key acceso al servicio de *Mandrill*
- **mandrill.webhook.key:** API Key utilización de webhooks de *Mandrill*
- **mandrill.webhook.url:** Endpoint expuesto para registro de eventos
- **logger.transports:** Configuración de los transportes que utiliza el logger

/server/config/environment/preproduction.js

```
// Preproduction specific configuration
// =====================================
module.exports = {
  // Server IP
  ip: process.env.OPENSHIFT_NODEJS_IP ||
    process.env.IP ||
    undefined,

  // Server port
  port: process.env.OPENSHIFT_NODEJS_PORT ||
    process.env.PORT ||
    8300,

  // MongoDB connection options
  mongo: {
    uri: process.env.MONGOLAB_URI ||
      process.env.MONGOHQ_URL ||
      process.env.OPENSHIFT_MONGODB_DB_URL + process.env.OPENSHIFT_APP_NAME
||
      'mongodb://localhost/baboon-preprod'
  },

  mail: {
    host: 'smtp.gmail.com',
    port: 465,
    from: '<10labs.test@gmail.com>',
    to: '<10labs.test@gmail.com>',
    subjectForgetPwd: 'Olvido contraseÒa Baboon',
    secure: true,
    auth: {
      user: '10labs.test@gmail.com',
      pass: 'XXXXXXXX'
    }
  },

  mandrill: {
    apikey: 'XXXXXXXX',
```

```
    webhook: {
      key: 'XXXXXXXX',
      url: 'http://server_url/api/webhook/event'
    }
  },

  logger: {
    transports: {
      file: {
        options: {
          filename: rootPath + '/logs/baboon.log',
          prettyPrint: true,
          json: false
        }
      }
    }
  }
};
```

Variables de utilidad para la configuración:

- **ip:** Url externa de acceso al servidor
- **port:** Puerto de acceso al servidor
- **mongo.uri:** Base de datos mongodb
- **mail.host:** Servidor de correo
- **mail.port:** Puerto del servidor de correo
- **mail.from:** Puerto del servidor
- **mail.to:** Destinatario del correo electrónico
- **mail.subjectForgetPwd:** Asunto de los correos de olvido de contraseña
- **mail.secure:** Opción de conexión segura
- **mail.auth.user:** Usuario de corre
- **mail.auth.pass:** Contraseña del usuario de correo
- **mandrill.apikey:** API Key acceso al servicio de *Mandrill*
- **mandrill.webhook.key:** API Key utilización de webhooks de *Mandrill*
- **mandrill.webhook.url:** Endpoint expuesto para registro de eventos
- **logger.transports:** Configuración de los transportes que utiliza el logger

/server/config/environment/production.js

```
// Production specific configuration
// =================================
module.exports = {
  // Server IP
  ip: process.env.OPENSHIFT_NODEJS_IP ||
    process.env.IP ||
    undefined,

  // Server port
  port: process.env.OPENSHIFT_NODEJS_PORT ||
    process.env.PORT ||
    8100,

  // MongoDB connection options
  mongo: {
    uri: process.env.MONGOLAB_URI ||
      process.env.MONGOHQ_URL ||
      process.env.OPENSHIFT_MONGODB_DB_URL + process.env.OPENSHIFT_APP_NAME
||
      'mongodb://localhost/baboon'
  },

  mail: {
    host: 'smtp.gmail.com',
    port: 465,
    from: '<10labs.test@gmail.com>',
    to: '<10labs.test@gmail.com>',
    subjectForgetPwd: 'Olvido contraseña Baboon',
    secure: true,
    auth: {
      user: '10labs.test@gmail.com',
      pass: 'XXXXXXXX'
    }
  },

  mandrill: {
    apikey: 'XXXXXXXX',
    webhook: {
```

```
      key: 'XXXXXXXX',
      url: 'http://server_url/api/webhook/event'
    }
  },

  logger: {
    transports: {
      file: {
        options: {
          filename: rootPath + '/logs/baboon.log',
          prettyPrint: true,
          json: false
        }
      }
    }
  }
};
```

Variables de utilidad para la configuración:

- **ip:** Url externa de acceso al servidor
- **port:** Puerto de acceso al servidor
- **mongo.uri:** Base de datos mongodb
- **mail.host:** Servidor de correo
- **mail.port:** Puerto del servidor de correo
- **mail.from:** Puerto del servidor
- **mail.to:** Destinatario del correo electrónico
- **mail.subjectForgetPwd:** Asunto de los correos de olvido de contraseña
- **mail.secure:** Opción de conexión segura
- **mail.auth.user:** Usuario de corre
- **mail.auth.pass:** Contraseña del usuario de correo
- **mandrill.apikey:** API Key acceso al servicio de *Mandrill*
- **mandrill.webhook.key:** API Key utilización de webhooks de *Mandrill*
- **mandrill.webhook.url:** Endpoint expuesto para registro de eventos
- **logger.transports:** Configuración de los transportes que utiliza el logger

/server/config/environment/test.js

```
// Test specific configuration
// ===========================
module.exports = {
  mongo: {
    uri: 'mongodb://localhost/baboon-test'
  },

  mail: {
    host: 'smtp.gmail.com',
    port: 465,
    from: '<10labs.test@gmail.com>',
    to: '<10labs.test@gmail.com>',
    subjectForgetPwd: 'Olvido contraseÒa Baboon',
    secure: true,
    auth: {
      user: '10labs.test@gmail.com',
      pass: 'XXXXXXXX'
    }
  },

  mandrill: {
    apikey: 'XXXXXXXX',
    webhook: {
      key: 'XXXXXXXX',
      url: 'http://server_url/api/webhook/event'
    }
  },

  logger: {
    transports: {
      file: {
        options: {
          filename: rootPath + '/logs/baboon.log',
          prettyPrint: true,
          json: false
        }
      }
    }
```

```
  }
};
```

- **mongo.uri:** Base de datos mongodb
- **mail.host:** Servidor de correo
- **mail.port:** Puerto del servidor de correo
- **mail.from:** Puerto del servidor
- **mail.to:** Destinatario del correo electrónico
- **mail.subjectForgetPwd:** Asunto de los correos de olvido de contraseña
- **mail.secure:** Opción de conexión segura
- **mail.auth.user:** Usuario de corre
- **mail.auth.pass:** Contraseña del usuario de correo
- **mandrill.apikey:** API Key acceso al servicio de *Mandrill*
- **mandrill.webhook.key:** API Key utilización de webhooks de *Mandrill*
- **mandrill.webhook.url:** Endpoint expuesto para registro de eventos
- **logger.transports:** Configuración de los transportes que utiliza el logger

Anexo XIII: Planificación inicial SPRINT-1

Duración SPRINT: 3 semanas (*inicio:* 14/09/15 – *fin:* 04/10/15)

Estimación – tiempo dedicación SPRINT-1:

Dedicación semanal: 27 horas

- *Imprevistos* → 4h
- *Desarrollo* → 27h – 4h (imprevistos) = 23h

Dedicación TOTAL – SPRINT: 27h/semana * 3semanas = 81 horas

- *Imprevistos* → 4h/semana * 3 semanas = 12h
- ***Desarrollo* → 81h – 12h (imprevistos) = 69h**

Estimación – Tareas SPRINT-1:

A cada estimación de una tarea, se le añadirán 2 horas extras como margen de error.

A cada tarea se le asignará un tamaño basado en *tallas* (S, M, L y XL) orientativo y representativo del tamaño/dificultad de la tarea con el fin de poder realizar una comparación entre las distintas tareas a simple vista y facilitar la posterior estimación.

API Campañas

Tarea	Talla	Estimación	TOTAL
Crear campaña	M	8h + 2h	**10h**
Editar campaña	S	4h + 2h	**6h**
Dar de baja campaña	S	3h + 2h	**5h**
Listar campañas	M	8h + 2h	**10h**
Lanzar campaña	XL	27h + 2h	**29h**
			62h

Registro de Eventos

Tarea	Talla	Estimación	TOTAL
Creación Webhook	S	3h + 2h	**5h**

~~Registro a nivel de campaña~~	~~L~~	~~12h + 2h~~	~~**14h**~~
~~Registro a nivel de correo electrónico~~	~~L~~	~~12h + 2h~~	~~**14h**~~
			33h

Analíticas Básicas

Tarea	Talla	Estimación	TOTAL
~~Analíticas a nivel de campaña~~	~~L~~	~~24h + 2h~~	~~**26h**~~
			26h

ESTIMACIÓN TOTAL (Tareas) = 62h + 33h + 26h = **121h** > 69h disponibles → reducir PILA DE TAREAS del SPRINT.

Anexo XIV: Replanificación SPRINT-1

Duración SPRINT: 2 semanas (*inicio:* 21/09/15 – *fin:* 04/10/15)

Estimación – tiempo dedicación SPRINT-1:

Dedicación semanal: 27 horas

- *Imprevistos* → 4h
- *Desarrollo* → 27h – 4h (imprevistos) = 23h

Dedicación TOTAL – SPRINT: 27h/semana * 2 semanas = 54 horas

- *Imprevistos* → 4h/semana * 2 semanas = 8h
- ***Desarrollo* → 54h – 8h (imprevistos) = 46h**

Estimación – Tareas SPRINT-1:

A continuación se muestra un desglose en subtareas de las principales funcionalidades a desarrollar en el Sprint tras su replanificación y la estimación de cada una de ellas.

S1_A1. Registrar aplicación

Subtarea	TOTAL
Frontend: formulario de creación	6 h
Backend: API registro + modelo datos	3 h
Testing	2 h
	11 h

S1_A2. Editar aplicación

Subtarea	TOTAL
Frontend: formulario de edición	6 h
Backend: API edición	3 h
Testing	2 h
	11 h

S1_A3. Eliminar/dar de baja aplicación

Subtarea	TOTAL
Frontend: método dar de baja	2 h
Backend: API dar de baja	1 h 30 m
Testing	2 h
	5 h 30 m

S1_A4. Listar aplicaciones registradas

Subtarea	TOTAL
Frontend: pantalla listado	10 h
Backend: API listado	5 h
Testing	3 h
	18 h

ESTIMACIÓN TOTAL (Tareas) = 11 h + 11 h + 5 h 30 m + 18 h = **45 h 30 m**

Anexo XV: Planificación SPRINT-2

Duración SPRINT: 2 semanas (*inicio: 12/10/15 – fin: 25*/10/15)

Estimación – tiempo dedicación SPRINT-2:

Dedicación semana 1: 17 horas

- *Imprevistos* → 4h
- *Desarrollo* → 17h – 4h (imprevistos) = 13h

Dedicación semana 2: 27 horas

- *Imprevistos* → 4h
- *Desarrollo* → 27h – 4h (imprevistos) = 23h

Dedicación TOTAL – SPRINT: 17h/semana1 + 27h/semana2 = 44 horas

- *Imprevistos* → 4h/semana * 2 semanas = 8h
- ***Desarrollo* → 44h – 8h (imprevistos) = 36h**

Estimación – Tareas SPRINT-2:

A continuación se muestra un desglose en subtareas de las principales funcionalidades a desarrollar en el Sprint y la estimación de cada una de ellas.

Tareas de desarrollo

S2_A1. Crear plantilla

Subtarea	**TOTAL**
Backend: API creación + modelo	6 h
Testing	3 h
	9 h

S2_A2. Listar plantillas

Subtarea	**TOTAL**
Backend: API listado	3 h
Testing	1 h 30 m
	4h 30 m

S3_A3. Ver detalles plantilla

Subtarea	TOTAL
Backend: API detalles	2 h 30 m
Testing	1 h 30 m
	4 h

S4_A4. Eliminar plantilla

Subtarea	TOTAL
Backend: API eliminado	2 h 30 m
Testing	1 h 30 m
	4 h

S4_A4. Editar plantilla

Subtarea	TOTAL
Backend: API edición	3 h 30 m
Testing	1 h 30 m
	5 h

S4_B1. Configurar servicio API Mandrill

Subtarea	TOTAL
Configuración entornos: API KEYs Mandrill	1 h
Wrapper NodeJS Mandrill (integración del servicio)	1 h 30m
	2h 30 m

Otras tareas

Tareas	TOTAL
Estudio API Mandrill: Templates	2 h
Diseño del Modelo de datos adecuado: - *Integración con Mandrill*: gestión de plantillas entre Mandrill y	3 h

Baboon - *Integración con Baboon*: módulo de aplicaciones	
	5 h

ESTIMACIÓN TOTAL (Tareas) = 9 h + 4 h 30 m + 4 h + 4 h + 5 h + 2 h 30 m + 5 h = **34 h**

Anexo XVI: Planificación SPRINT-3

Duración SPRINT: 3 semanas (*inicio: 21/12/15 – fin: 11/01/16*)

Estimación – tiempo dedicación SPRINT-3:

Dedicación semana 1: 16 horas

- *Imprevistos* → 4h
- *Desarrollo* → 16h – 4h (imprevistos) = 12h

Dedicación semana 2: 14 horas

- *Imprevistos* → 4h
- *Desarrollo* → 14h – 4h (imprevistos) = 10h

Dedicación semana 3: 20 horas

- *Imprevistos* → 4h
- *Desarrollo* → 20h – 4h (imprevistos) = 16h

Dedicación TOTAL – SPRINT: 16h/semana1 + 14h/semana2 + 20h/semana3 = 50 horas

- *Imprevistos* → 4h/semana * 3 semanas = 12h
- ***Desarrollo* → 50h – 12h (imprevistos) = 38h**

Estimación – Tareas SPRINT-3:

A continuación se muestra un desglose en subtareas de las principales funcionalidades a desarrollar en el Sprint y la estimación de cada una de ellas.

Tareas de desarrollo

S3_A1. Crear campaña

Subtarea	TOTAL
Backend: API creación + modelo	6 h
Testing	3 h
	9 h

S3_A2. Listar campaña

Subtarea	TOTAL

Backend: API listado	2 h 30 m
Testing	1 h 30 m
	4h

S3_A3. Ver detalles campaña

Subtarea	**TOTAL**
Backend: API detalles	2 h 30 m
Testing	1 h 30 m
	4 h

S3_A4. Eliminar campaña

Subtarea	**TOTAL**
Backend: API eliminado	1 h 30 m
Testing	1 h 30 m
	3 h

S3_A5. Editar campaña

Subtarea	**TOTAL**
Backend: API edición	4h
Testing	1 h 30 m
	5 h 30 m

S3_A6. Lanzar campaña

Subtarea	**TOTAL**
Especificación y documentación: endpoint externo	2 h
Backend: API lanzamiento	4 h
Testing	1h 30 m
	7h 30 m

Otras tareas

Tareas	TOTAL
Análisis y Diseño: Módulo Campañas	8 h
	8 h

ESTIMACIÓN TOTAL (Tareas) = 9 h + 4 h + 4 h + 3 h + 5 h 30 m + 7 h 30 m + 8 h = **41 h**

Anexo XVII: Planificación SPRINT-4

Duración SPRINT: 2 semanas (*inicio: 18/01/16 – fin: 01/02/16*)

Estimación – tiempo dedicación SPRINT-4:

Dedicación semana 1: 16 horas

- *Imprevistos* → 4h
- *Desarrollo* → 16h – 4h (imprevistos) = 12h

Dedicación semana 2: 14 horas

- *Imprevistos* → 4h
- *Desarrollo* → 14h – 4h (imprevistos) = 10h

Dedicación TOTAL – SPRINT: 16h/semana1 + 14h/semana2 = 50 horas

- *Imprevistos* → 4h/semana * 2 semanas = 12h
- ***Desarrollo* → 50h – 12h (imprevistos) = 38h**

Estimación – Tareas SPRINT-4:

A continuación se muestra un desglose en subtareas de las principales funcionalidades a desarrollar en el Sprint y la estimación de cada una de ellas.

Tareas de desarrollo

S4_A1. Servicio externo

Subtarea	TOTAL
Módulo Aplicaciones: Backend	6 h
Módulo Aplicaciones: Frontend	
Módulo Campañas: petición HTTP a Servicio Externo	3 h
	9 h

S4_A2. Registro de eventos Mandrill

Subtarea	TOTAL
Creación Webhooks	2 h 30 m
Securización Webhooks	

Registro de eventos en el sistema	1 h 30 m
	4h

ESTIMACIÓN TOTAL (Tareas) = 9 h + 4 h + 4 h + 3 h + 5 h 30 m + 7 h 30 m + 8 h = **41 h**