eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado de Ingeniería Informática

▪ **T**rabajo **F**in de **G**rado ▪

Ingeniería del Software

MediaKaanª: plataforma web de donación e intercambio multimedia.

Miguel Ángel Blanco Fernández

Junio 2020

Dirección

José Ángel Vadillo Zorita

Agradecimientos

Antes de proceder con la memoria, me gustaría agradecer a todas las personas que han permanecido conmigo y he sentido muy de cerca en esta gran etapa que he vivido, que será finalizada con este proyecto.

Primeramente doy gracias a mi familia, por todo el apoyo recibido y todo lo que han trabajado duro para que yo pudiera elegir, estudiar y terminar este grado.

Quiero agradecer a mi pareja Marta, que a pesar de la distancia que nos separa ha estado acompañándome en los buenos momentos y los no tan buenos durante toda la carrera, ofreciéndome siempre su apoyo mientras me escuchaba y me daba fuerzas para continuar.

Me gustaría dar las gracias a los amigos que tengo cerca a Iker, a Nerea y a Julen, por todos los ánimos dados y todos los fines de semana en los que habéis hecho que fuera más fácil volver a estudiar a lo largo de la semana. Y por otra parte a los que tengo lejos, a Elizabeth, a Rocío, a Ruby, a Sergio, a Cristian, a José Manuel, a Paloma y a Agustín, cuya presencia he sentido muy cerca y también se han preocupado por mí dándome ánimos en aquellos momentos complicados del grado.

También quiero agradecer a aquella gente que he conocido durante el grado. A mi director de proyecto José Ángel que me ha ayudado y apoyado con este proyecto para llevarlo a cabo a pesar de todas sus complicaciones, habiéndome además introducido y enseñado parte de las tecnologías que aquí utilizo gracias a sus asignaturas como profesor. Y también gracias a aquellos compañeros o profesores que me han ayudado durante el grado a formarme personalmente y como profesional, en especial a Iñaki, a Josemi, a Montse, a Ferran, a Mikel, a Andrea, a Oier, a Alfonso, a Miguel, a Mai, a Xabi, a Iñigo y a otros tantos que igual no llego a recordar en este momento.

Ha sido un viaje muy difícil y sin todos ellos probablemente no habría llegado a donde estoy ahora realizando este proyecto, gracias de todo corazón a todos aquellos que directamente o indirectamente habéis hecho todo esto posible.

Índice

Índice de figuras y tablas

Figuras

Tablas

1. Introducción

MediaKaanª es una aplicación web para donar, intercambiar o compartir entre usuarios artículos multimedia como películas, música, cómics, libros y videojuegos. Este capítulo describirá de donde surge la idea principal del proyecto teniendo como objeto esta aplicación e introduciremos los diferentes capítulos que nos encontraremos a lo largo de la memoria sobre cómo ha sido su desarrollo, cuál ha sido su gestión y qué tecnologías se han utilizado.

La idea principal de esta aplicación web multiplataforma surge a partir de la segunda semana de Febrero tras cuestionarme durante unos días de qué iba a realizar mi TFG. Gran parte de mi tiempo libre lo dedico al entretenimiento consumiendo multimedia en digital y también en físico leyendo cómics o jugando en consolas viejas, pero al fin y al cabo lo hago más en digital. Y es gracias a esto que surgió dicha idea, estos años y los siguientes que vienen se ve una **predisposición a que la multimedia en digital acabe por sustituir enteramente a lo físico** y ante esta situación poder crear un **espacio** para el contenido **multimedia en físico redistribuido de forma libre y desinteresada** me pareció una buena premisa de la que hacer una aplicación.

Durante los primeros capítulos **Antecedentes** y **Objetivos y Alcance del proyecto** se puede comprobar con qué conocimientos y estado se partió para su desarrollo y cuáles eran las metas a lograr para obtener una funcionalidad básica de lo esencial de la aplicación, además del radio que tomaría dentro del proyecto del que forma parte junto a los demás paquetes de trabajo que lo componen.

En cuanto al diseño de la solución adoptada establecida con las tecnologías elegidas tendremos los capítulos siguientes **Análisis y diseño**, **Tecnologías** e **Implementación**. **Análisis y diseño** servirá para conocer el modelo de dominio no relacional que se ha establecido para la aplicación junto al modelo de casos de usos con el funcionamiento interno de cada uno de ellos detallado bajo la arquitectura MTV[1] de Django. Por otra parte, el capítulo dedicado a las **tecnologías** servirá para repasar las razones y ventajas que existen al haber elegido las principales tecnologías del proyecto que son **Django**, **MongoDB**, **JQuery** y **Bootstrap**. Por último, describiré los aspectos de **implementación** donde se profundizarán los resultados descritos en el capítulo de **Análisis y diseño** comentando la estructura del proyecto, su arquitectura y de forma detallada la implementación de uno de los casos de uso.

Y para terminar tendremos los capítulos finales **Gestión y seguimiento** y **Conclusiones**, donde hablaremos sobre las metodologías que se han seguido, cómo se ha realizado la monitorización y control del proyecto y cuáles han sido las reflexiones finales acerca del mismo.

[1] Modelo Template View, una arquitectura similar a la conocida Modelo Vista Controlador.

2. Antecedentes

En este capítulo se expondrán los antecedentes del proyecto donde se visualizará el interés inicial que había en el área de conocimientos a tratar, los tipos de tecnologías encontradas para investigar para la aplicación y un análisis de las aplicaciones similares a la nuestra para encontrar aspectos sobre los que destacar.

2.1. Interés previo al proyecto

Para la concepción de este proyecto, antes de que surgiera su idea principal hubo una cosa clara y esa era que la aplicación a desarrollar fuera una aplicación web. A lo largo de la carrera y de la especialidad, este es un tema que se llega a trabajar en al menos tres o cuatro asignaturas y de las que se obtienen ciertos conocimientos generales y unas bases con las que poder especializarse adecuadamente si se investiga sobre los mismos conceptos tratados en estos cursos.

La idea de la portabilidad que ofrecen las aplicaciones web frente a otros tipos es el pilar principal sobre el que se apoya mi interés, el hecho de que algo funcione desde distintos dispositivos con el mismo código. Y de forma paralela al grado este fue un tema que me motivó a investigar por mi cuenta, llegando a realizar algunos prototipos de pequeños proyectos que abandoné sobre interfaces responsive y llegando a asistir a defensas de trabajos de fin de máster relacionados con el área en la Universidad Politécnica de Madrid, después de que mi padre me animara a asistir al suyo. Lo que acabó dando como resultado que mi interés por desarrollar algo sobre este área permeara en algo más grande como es la realización del TFG.

2.2. Aplicación a desarrollar y motivaciones

Aprovechando la idea de la portabilidad comentada anteriormente, surgió la idea de realizar una aplicación multiplataforma apoyada en la difusión libre de la cultura y educación audiovisual. Algo que funcionara como un **sitio de dar y dejar** frente a las aplicaciones que funcionaban como un **mercado de segunda mano**.

Y debido a que esta aplicación parecía requerirlo, se decidió estudiar tecnologías concretas que se conocían brevemente para motivar el aprendizaje de algo más novedoso sobre el desarrollo de aplicaciones web.

Esta aplicación requería que el diseño de su interfaz fuera adaptable a las pantallas de los dispositivos por lo que había que estudiar sobre herramientas de diseño responsive alejadas del diseño CSS que se había conocido en el grado. La escalabilidad de datos para el tipo de aplicación que pensábamos desarrollar también era algo a estudiar por lo que hacía falta investigar sobre el diseño y los sistemas de las bases de datos no relacionales, modelos diferentes a lo que se había estudiado en el grado y que se utilizaban en gran parte de las aplicaciones modernas. Y por último, estaba la elección de las herramientas de desarrollo web, que dadas las dimensiones de la aplicación hacía necesario escoger una de alto nivel (framework) frente a las de bajo nivel (PHP,

JS, PhpMyadmin, WAMP, CSS...) que se habían visto de forma más profunda en el grado y la especialidad.

2.3. Análisis de aplicaciones similares

Desarrollar una aplicación de estas características precisaba de un estudio de aplicaciones similares para encontrar aquellos puntos sobre los que destacar y aprovechar con respecto a la idea principal de la misma. Inicialmente, se pensó en investigar sobre aquellas aplicaciones que tuvieran en cierto modo la misma idea principal pero debido a que no se encontraron resultados de aplicaciones de este tipo, se acabó optando por investigar sobre las más parecidas a nuestra idea, las de compra y venta de segunda mano.

El elemento a destacar aquí era el componente social debido a que los usuarios de la aplicación **intercambiarían, compartirían o donarían** el contenido que ellos presentarán en la propia aplicación. Para ello las aplicaciones de las que se hizo un análisis de funcionamiento fueron **Milanuncios** y **Wallapop**, las más populares en lo que segunda mano se refiere.

- **Milanuncios:** Esta aplicación es la más popular en segunda mano y reúne todo tipo de contenido que se puede vender o comprar pasando por inmobiliaria, mascotas, ocio, deportes, etc. La presentación de esta aplicación es correcta y la gran cantidad de tipos de cosas que se pueden encontrar anima a realizar búsquedas de contenido con filtros específicos, sin embargo su gran contra estaría en hacer poco personal la forma en la que se ofrecen los objetos debido a que su propósito principal es el de anunciar de forma correcta e impersonal las cosas para luego contactar con su propietario.

- **Wallapop:** Dentro de la compra-venta de segunda mano, es la segunda aplicación más popular reuniendo al igual que la aplicación anterior todo tipo de contenido. El punto de esta aplicación que más se podría destacar es que visualizando la presentación de cada artículo se puede apreciar como se hace más personal y se pueden visualizar datos o opiniones acerca del propietario de forma más clara. Seguido de esto también estarían los canales de comunicación, que serían más inmediatos al poder abrir conversación por medio de un botón y por otra parte la posibilidad de guardar objetos en favoritos. En cuanto a sus puntos mejorables, se podría decir que respecto a nuestra aplicación, aunque la información de los usuarios se muestre adecuadamente puede que sea mejor la inclusión de más cómo por ejemplo una descripción breve del usuario.

Tras el análisis final de los pros y contras de estas aplicaciones se llegó a la conclusión de lo que se quería conseguir, **Wallapop** ofrecía un plus sobre hacer más cercana la compra o la venta de artículos de segunda mano y algo más próximo a la idea que nosotros queríamos conseguir. Fijándonos en ello y en nuestro tipo de aplicación, se decidió que a pesar de que nuestra aplicación pudiera ser sobre donación multimedia con artículos sería importante motivar más el apartado social e incentivar las acciones desinteresadas para destacar sobre las aplicaciones similares que habíamos visto.

3. Objetivos y Alcance del proyecto

Este capítulo se centrará en describir en su primera sección los objetivos que forman el trabajo de fin de grado, empezando por aquellos generales para desarrollar el proyecto, repasando aquellos más específicos y finalizando con aquellos más concretos con los que se planea que la aplicación en su versión final satisfaga. Finalmente, a través de su siguiente sección, se especificará el alcance total del proyecto por medio de una estructura de desglose de trabajo explicando brevemente los paquetes de trabajo que la conforman.

3.1. Objetivos

Los objetivos principales para el desarrollo de este proyecto serán los siguientes.

- Desarrollar una aplicación web multiplataforma.
- Pruebas de la aplicación en local.
- Decidir servicio de hosting.
- Despliegue en la nube de la aplicación y pruebas en la nube.
- Depuración y alojamiento en la nube.

En relación a las tecnologías elegidas los objetivos específicos que se necesitarán satisfacer para el proyecto serán estos.

- Diseño de una base de datos NoSQL y desarrollo con MongoDB.
- Diseño de la interfaz multiplataforma con Bootstrap.
- Integración de la interfaz gráfica y desarrollo de la lógica de la aplicación con Django.

De forma más concreta, la aplicación desarrollada en su versión final deberá al menos responder a los siguientes objetivos.

1. El objetivo principal del proyecto será crear una plataforma web responsive albergada por usuarios que se compartan, se intercambien o se den **material multimedia en físico** entre ellos. El material que podrá poseer la aplicación irá desde: libros, música, videojuegos, películas, series y cómics.
2. El usuario de la aplicación podrá registrarse, loguearse, consultar información acerca del material multimedia que esté subido a la aplicación por otros usuarios, subir/editar/asignar su propio material para otros usuarios, consultar/editar su perfil, valorar/consultar otros perfiles de usuario y escribir a otro usuario por medio de chat para intentar conseguir un material que alguien haya puesto en la aplicación.
3. La información de los usuarios estará compuesta de una foto, un nick, una contraseña, su email, una descripción breve de él, una ubicación, la fecha de creación del usuario, sus valoraciones y el año de nacimiento.

4. La ubicación de los usuarios está pensada para ser a ámbito nacional, por lo que en edad temprana de la aplicación tendrá asignada una comunidad autónoma, una provincia y la localidad correspondiente.
5. El material multimedia que registren los usuarios estará compuesta por un identificador, un nombre, una foto, una descripción, su categoría (**película, videojuego, comic...**), la acción que se quiere hacer con ello (**intercambio, comparto, dono**), si está o no asignado, su fecha de adición y los tags.
6. La aplicación tendrá tres roles para el usuario, uno que será el de administrador, el otro que será el de usuario normal (cliente) que utilizará la aplicación y otro para el usuario que no se haya registrado o iniciado sesión, que podrá consultar los artículos.
7. El usuario de tipo administrador podrá gestionar cuentas y artículos, añadiéndolos, consultándolos o eliminándolos. Servirá para hacer pruebas en el desarrollo.
8. Para cada objeto multimedia habrá varios chats de la gente interesada, que establecerán comunicación entre el usuario interesado y el que ha puesto el objeto. Estará identificado por un id y tendrá una fecha de creación, dichos chats tendrán un conjunto de los mensajes entre los dos usuarios que tendrán una id de mensaje, el contenido y su fecha y hora correspondientes.
9. Los elementos constantes que aparecerán en la aplicación web en sus diversas páginas serán los siguientes en orden descendente:
    a. Una franja con registrarse e iniciar sesión a la derecha, si el usuario está logueado se le mostrará su nombre de perfil ahí donde si hace click podrá editarlo o consultarlo además de un botón de subir para añadir a la aplicación material multimedia.
    b. Una franja con el logo de la aplicación seguida de una barra de búsqueda donde buscar por tags el material multimedia.
    c. Una franja seguida con los apartados de: Inicio, chats y acerca de.
10. La página de “inicio” mostrará únicamente el número de cosas que hay compartidas por los usuarios, seguidas por las categorías de dichas cosas con su cantidad correspondiente concreta.
11. La página de “chats” mostrará los chats del usuario que tiene con los usuarios que están compartiendo un objeto en concreto. (estos no irán en tiempo real)
12. La página de “acerca de” mostrará información de cómo funciona la página, quien es el autor de la app, cómo contactarlo y un formulario para enviar recomendaciones para la aplicación.
13. Cuando el usuario busque por tags se le mostrará una lista de objetos encontrados, pudiendo acceder a la información de ellos. El usuario podrá filtrar la búsqueda por fecha (ascendente, descendente) y lugar (especificado).
14. La plataforma web deberá contar con un sistema de seguridad que prevenga las entradas de datos maliciosas (inyecciones de código del SGBD, scripts malintencionados...) y sanitice los datos antes de enviarlos al servidor.
15. Las contraseñas de los usuarios deberán seguir una serie de reglas y deberán ser guardadas de forma segura y cifrada en la base de datos de la aplicación.
16. El servicio de hosting elegido para la aplicación desplegada en la nube deberá ser de calidad y deberá garantizar varias utilidades de seguridad. Entre ellas, deberá dotar al servidor donde esté alojada de un certificado SSL para

garantizar la seguridad en la comunicación entre usuario y servidor en la aplicación. Y por otra parte, deberá garantizar un sistema de acceso seguro para realizar pruebas de la aplicación en su versión de prelanzamiento en la nube sin que quede accesible para los usuarios finales.

Aunque estos sean objetivos específicos y requisitos de la aplicación, muchas de las funcionalidades podrán variar conforme se desarrolle la aplicación.

3.2. Alcance del proyecto

El alcance de nuestro proyecto lo describiremos a partir de una estructura de desglose de trabajo (EDT) que será un esquema que estructurará los paquetes de trabajo referentes al proyecto. Nuestro EDT tendrá la siguiente organización.

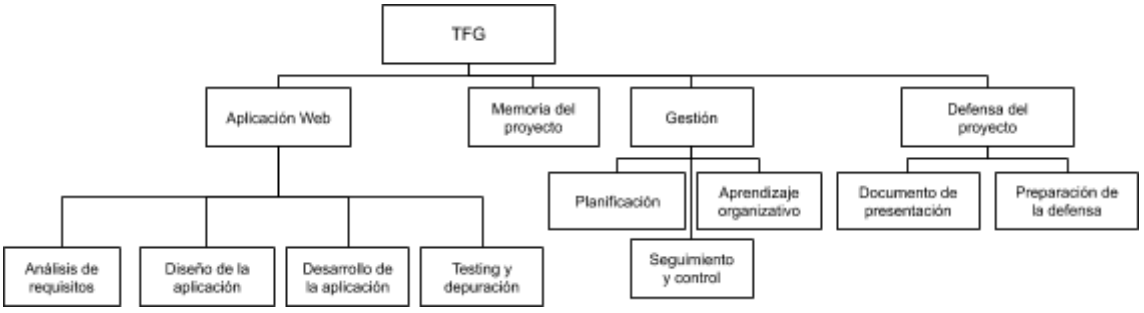


Figura 1. Esquema de Desglose de Trabajo del proyecto

Los principales paquetes de trabajo irán preparándose de manera simultánea y reunirán el siguiente contenido:

- **Aplicación web:** Este paquete cubrirá todo lo relacionado con la aplicación comenzando por su análisis de requisitos con su modelo de dominio y su modelo de casos de uso, el diseño de todo lo que la completa, el desarrollo de esta y sus pertinentes pruebas y depuración conforme se vaya desarrollando.
- **Memoria del proyecto:** Este paquete de trabajo soportará todo lo referente a la realización de la memoria del proyecto.
- **Gestión:** Cubrirá el trabajo que se realice para la planificación del proyecto, el seguimiento y control respecto a esta y el aprendizaje organizativo para llevarlo a cabo adquiriendo los conocimientos necesarios de cada tecnología.
- **Defensa del proyecto:** De cara a defender el proyecto, este paquete cubrirá la parte referente a la preparación de la presentación y la preparación de la defensa antes de llevarla a cabo.

4. Análisis y diseño

En este capítulo realizaremos una visión global sobre aquellos casos de uso que forman nuestra aplicación entendiendo el flujo de los mismos respecto a la arquitectura de los proyectos de Django y el diseño del modelo de dominio escogido para una base de datos no relacional en MongoDB. Finalmente, presentaré en más detalle el diseño de un caso de uso mediante un diagrama de secuencia bajo el patrón de arquitectura Django.

4.1. Modelo de casos de uso

La aplicación según el análisis de requisitos realizado tendría el siguiente modelo de casos de uso. En este apartado se verán los papeles que desempeñan en la aplicación los tres roles (Invitado, Usuario y Administrador) junto a aquellos casos de uso que los componen.

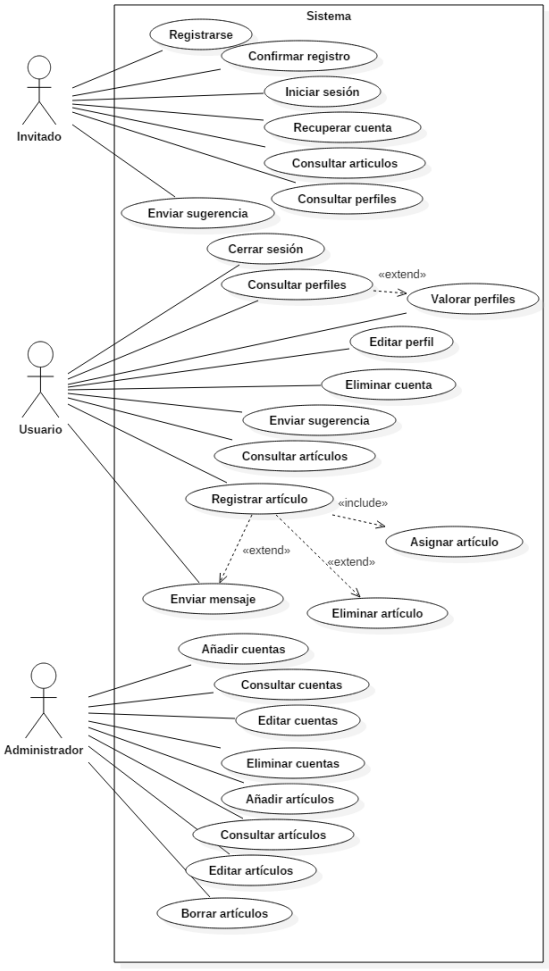


Figura 2. Modelo de casos de uso de la aplicación

4.1.1. Rol de invitado

El rol de invitado corresponderá a la persona que requiera información pero sin llegar a realizar intercambios, donaciones o publicaciones de artículos. Un invitado podrá consultar artículos de la aplicación o bien pasar al rol usuario (con sus casos de uso correspondientes) haciendo uso del caso registrarse, confirmar registro e iniciar sesión. Si el invitado disponía de un usuario anteriormente y ha olvidado la contraseña, podrá utilizar recuperar cuenta para poder volver a iniciar sesión y pasar a los casos de uso del rol usuario.

- **Registrarse:** Será el caso de uso con el que un invitado se cree una cuenta para pasar a ser usuario iniciando sesión posteriormente. El invitado a través de la plantilla de registro introducirá una serie de datos obligatorios ayudado por ciertos textos sobre cómo rellenar cada campo. Esta serie de datos estará compuesta por su nombre de usuario, su email, su contraseña, una pequeña descripción de él, su foto de usuario y su ubicación, sujetos todos ellos a validaciones. Cuando estos datos sean enviados, una vista procesará dicha petición comprobando si el formulario es válido. En caso negativo al invitado se le devolverá a la plantilla de registro junto a los errores de cada campo que haya rellenado mal. Mientras que en caso positivo a partir de esos datos se crearán los objetos correspondientes a la información del usuario en la base de datos, enviando un correo al invitado para que confirme su registro como usuario e informándole por medio de la plantilla de que el registro ha sido exitoso.

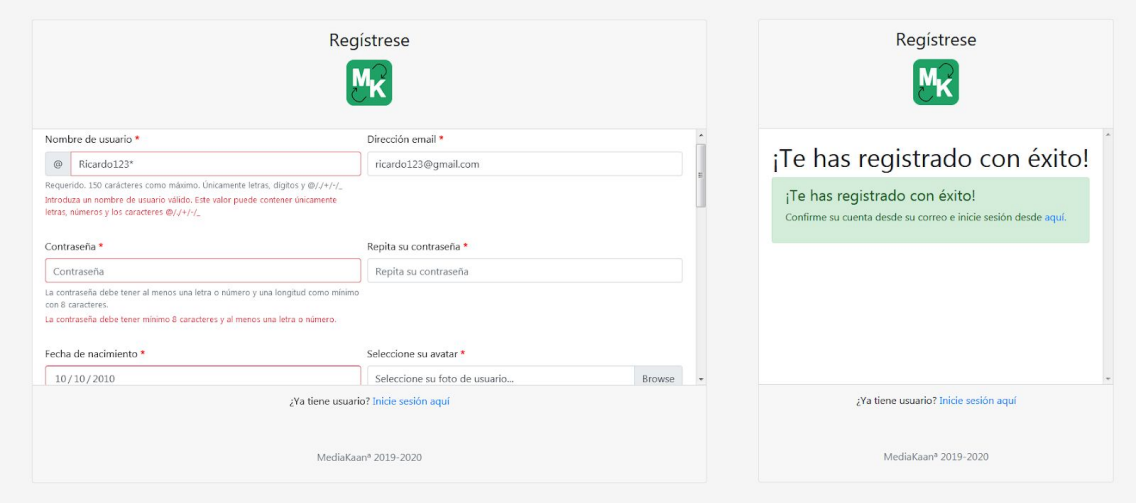


Figura 3. Interfaz de registro tras registrarse con errores y sin errores

- **Confirmar cuenta:** Un invitado que ya haya registrado su cuenta de usuario recibirá en su correo electrónico un email con un enlace único compuesto por su código de usuario y su código de activación que estará codificado con una marca de tiempo, el atributo booleano de activación del objeto usuario y su código de usuario. Una vez el invitado clique en el enlace, la vista de confirmar cuenta tomará de la base de datos el objeto usuario correspondiente al código de usuario del enlace y se comprobará si el código de activación es válido junto a dicho objeto. Si la vista determina que la confirmación ha sido correcta la variable de activación del objeto usuario se evaluará a verdadero, se actualizará dicho objeto en la base de datos y se le mostrará al invitado

la plantilla de activación con un mensaje de éxito. En el caso contrario dicha vista procesará la plantilla de activación con un mensaje de error para informar al invitado de que la activación de su cuenta de usuario no se ha producido.

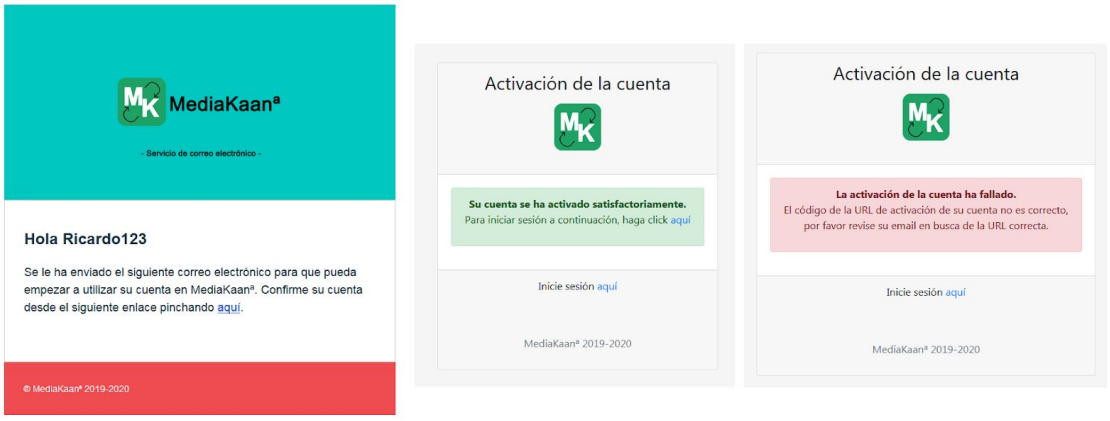


Figura 4. Correo e interfaz de activación con su mensaje de éxito y de error

- **Iniciar sesión:** Un invitado podrá identificarse a través de la aplicación con este caso de uso (siempre que ya se haya registrado y confirmado su cuenta) introduciendo su nombre de usuario y contraseña en la plantilla de inicio de sesión. Una vez estos datos sean enviados, la vista comprobará en la base de datos si el usuario proporcionado es correcto y está activo. Si esta comprobación es positiva la vista guardará en la petición de manera persistente el objeto usuario correspondiente y le redirigirá al invitado como usuario a la página principal. Si por lo contrario esta es negativa, la vista informará al usuario a través de la plantilla de inicio de sesión de que no ha confirmado su cuenta de usuario o de que los datos identificativos del usuario introducidos son incorrectos.

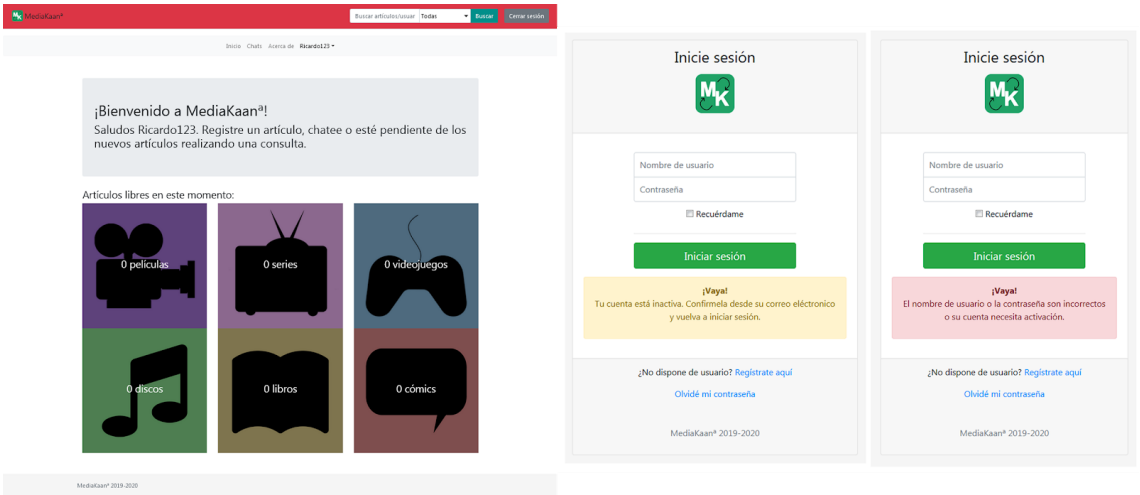


Figura 5. Página índice con un inicio de sesión correcto e interfaz de inicio de sesión con errores

- **Recuperar cuenta:** En caso de que el invitado haya olvidado la contraseña de su cuenta de usuario, este caso de uso le permitirá recuperar su cuenta ingresando su email en el formulario de la plantilla de olvido de contraseña. La vista una vez este

dato sea enviado tomará el usuario correspondiente a dicho email si existe y le enviará un correo electrónico con un enlace formado por el código del usuario y su código de recuperación (que se codifica de la misma manera que el código de activación de la cuenta).

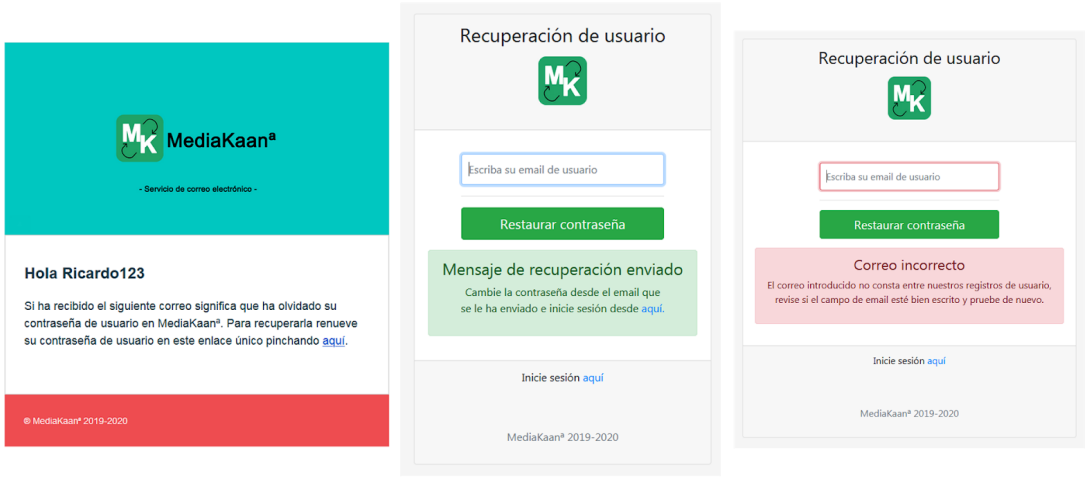


Figura 6. Correo de recuperación e interfaz de recuperación sin errores y con errores

Si el invitado procede a clicar en dicho enlace la vista de cambio de contraseña renderizará la plantilla de cambio de contraseña donde podrá ingresar su contraseña y su réplica. La vista de cambio de contraseña detectará que se han enviado datos y comprobando que el formulario es correcto junto a los códigos, actualizará el objeto usuario tomado con el código de usuario con la nueva contraseña mostrando al invitado la plantilla con un mensaje de éxito. Si por el contrario, la contraseña no cumple las validaciones o los códigos son incorrectos, la vista renderizará la plantilla con mensajes de error.

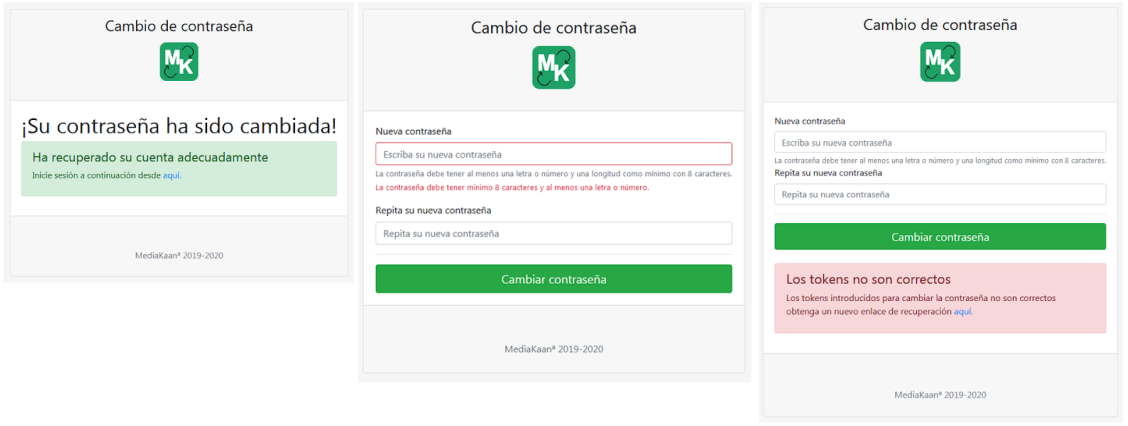


Figura 7. Interfaz de cambio de contraseña sin errores y con errores

- **Consultar artículos:** El invitado podrá consultar los artículos que haya en la aplicación con un filtro de categoría (películas, música, cómics...). Este podrá hacerlo desde la plantilla índice de la aplicación clicando en cualquier categoría con artículos o desde cualquier plantilla de la aplicación seleccionando en la parte superior la

categoría desde una lista desplegable y escribiendo en el campo de búsqueda una consulta con tags (separados por comas) o una aproximación del título de un artículo. Una vez decida buscar, la vista de consulta de artículos realizará una consulta con rango de resultados en la base de datos según la categoría, si no están asignados a otros usuarios y si se ha hecho o no con tags la búsqueda, devolviendo la información de diez de ellos por página en la plantilla de consulta de artículos para no realizar una carga excesiva de resultados y aprovechar mejor los recursos de la base de datos. Los artículos mostrados por otra parte mostrarán información básica que también ayudará a realizar otro tipo de consultas, si el invitado clica en un nombre de un dueño podrá ver la información de su perfil y si clica en uno de los tags de los artículos la misma vista de consulta de artículos hará una consulta por tag tal y como se ha comentado.

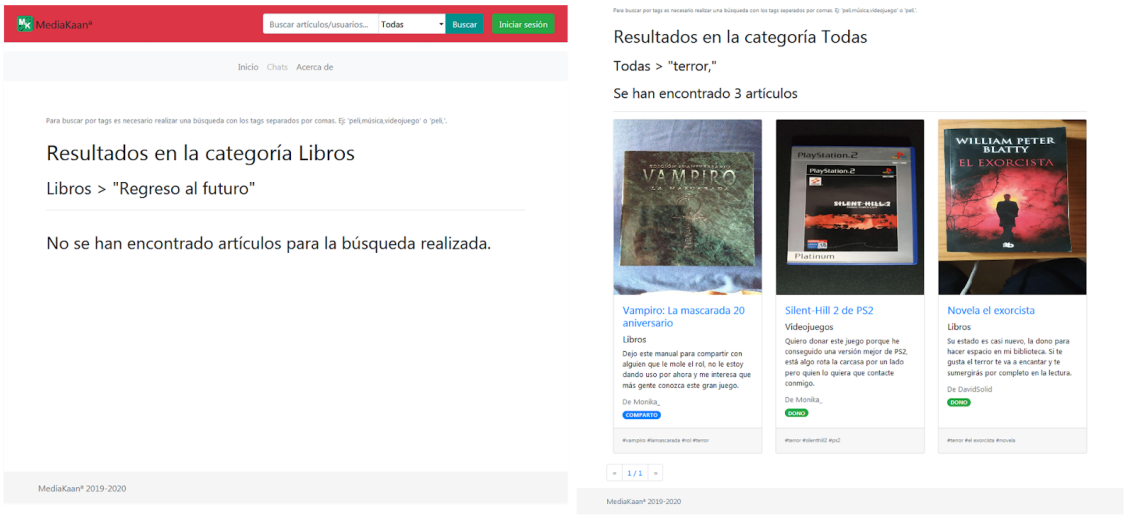


Figura 8. Interfaz de búsqueda de artículos sin resultados y con resultados

Si el invitado decide consultar la información de uno de los artículos de manera individual, la vista asociada consultará a la base de datos por el identificador del artículo y devolverá a la plantilla correspondiente los datos del propietario del artículo con su ubicación junto a la información del artículo correspondiente al identificador. La información del artículo que se podrá ver será su título, su descripción, la acción que desea el propietario hacer con él (DONO, COMPARTO, INTERCAMBIO), su categoría, sus etiquetas y una imagen del propio artículo. En ambos casos si las plantillas no han encontrado artículos en la aplicación según las consultas, se informará de esto al usuario a través de un mensaje.

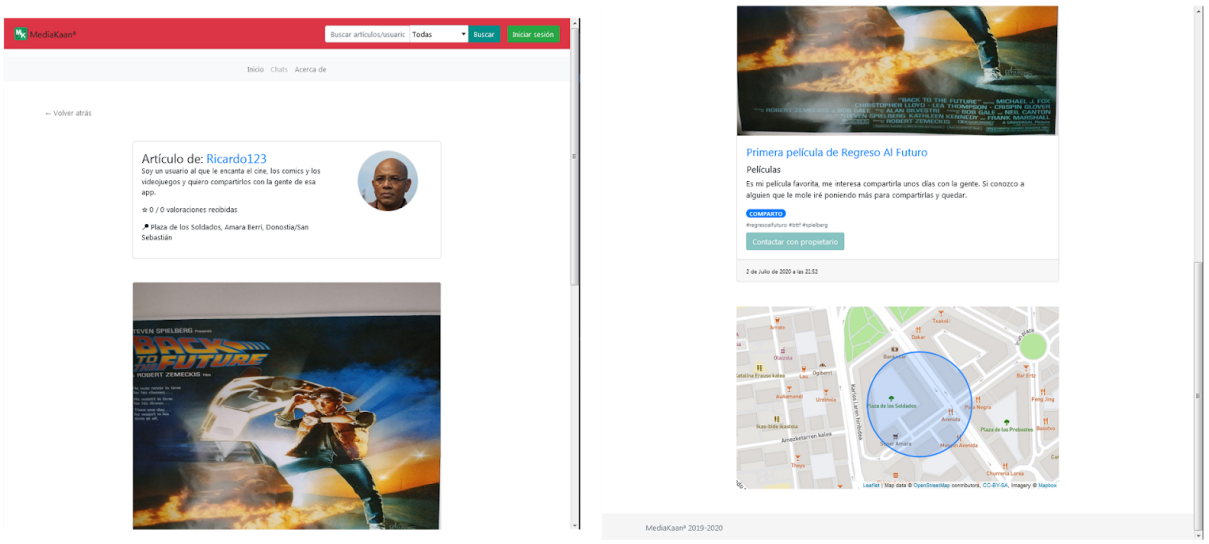


Figura 9. Interfaz de artículo individual con su información

- **Consultar perfiles:** Si el invitado a la hora de realizar una búsqueda en cualquier plantilla selecciona desde los mismos controles de consulta de artículos la categoría usuarios, podrá consultarlos escribiendo una aproximación del nombre de usuario que quiere encontrar. Producido el envío de información, la propia vista de consulta de artículos procesará dicha petición detectando que la categoría es usuarios y reenviará la petición a la vista de consulta de perfiles de usuarios con los datos de consulta. La vista de consulta de perfiles internamente comprobará los datos enviados y si los hay procederá a realizar una consulta a la base de datos con un rango de diez resultados establecido por número de página, que inicialmente será la primera y tendrá controles para que el usuario no exceda dichos límites. Recogidos los objetos de la base de datos, la plantilla de consulta de perfiles los mostrará dando información básica de cada uno.

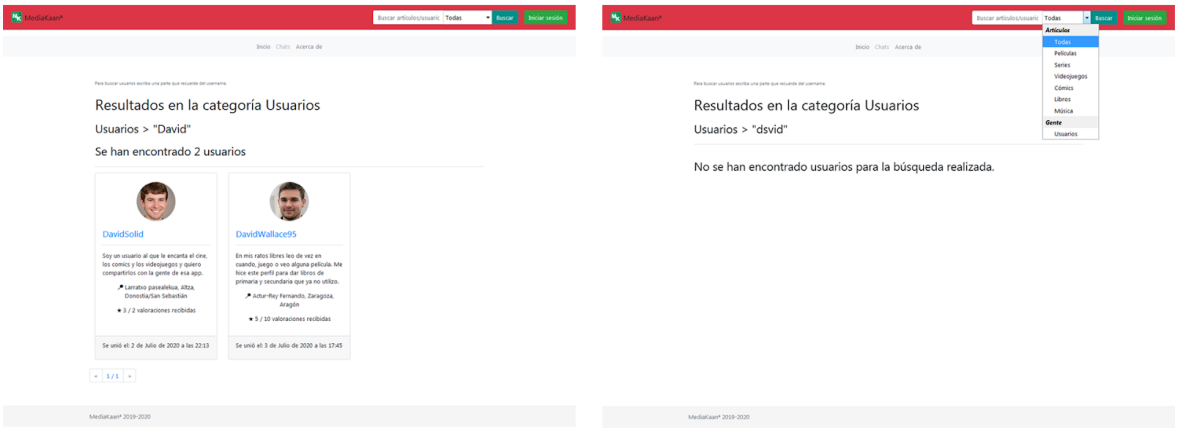


Figura 10. Interfaz de búsqueda de usuarios con resultados y sin resultados

Si en este punto el invitado decide consultar un usuario individual, la vista de consulta de perfiles individual internamente consultará a la base de datos los datos de un usuario que correspondan con el nombre de usuario seleccionado y devolverá de él en

su plantilla correspondiente su información de perfil, su ubicación y los artículos disponibles que tiene subidos a la aplicación.

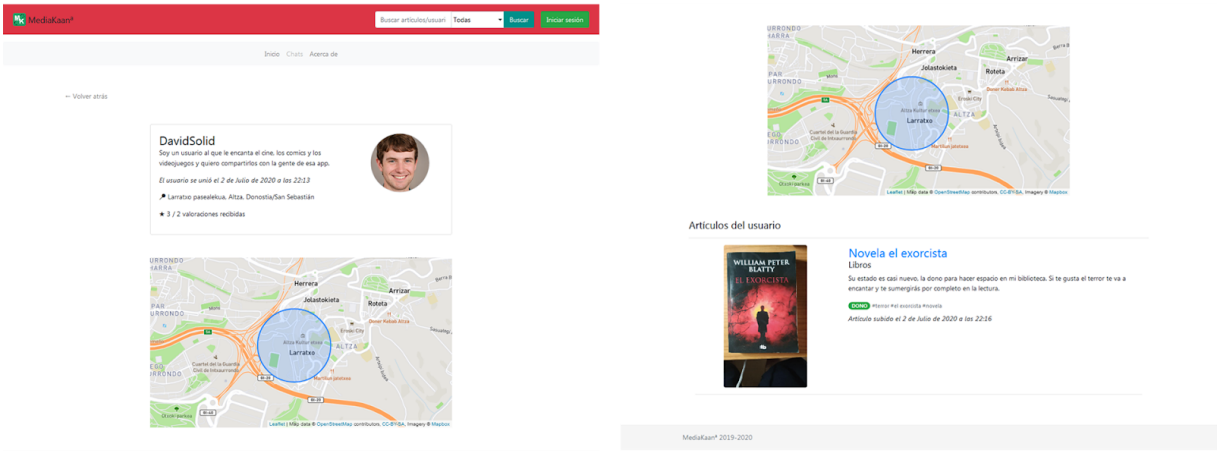


Figura 11. Interfaz de usuario individual con su información y sus artículos

- **Enviar sugerencia:** Un invitado que quiera sugerir una idea o mejora para la aplicación podrá rellenar desde la plantilla de *Acerca de* un formulario con su mensaje. La vista de *Acerca de* procesará dicha petición después de validar que hay texto y son suficientes caracteres, enviará al desarrollador de la aplicación por email un mensaje anónimo con la sugerencia y a través de la plantilla mostrará un mensaje de éxito. En caso de errores de validación, la plantilla mostrará un mensaje de error.

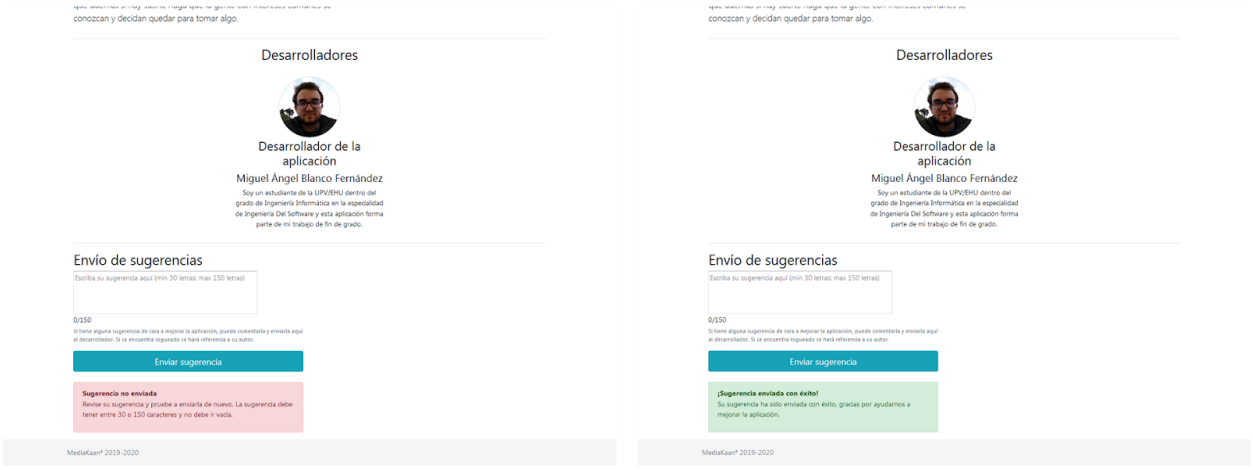


Figura 12. Interfaz de Acerca de con envío de sugerencia incorrecto y correcto

4.1.2. Rol de usuario

El rol de usuario será aquel que utilice los casos de uso principales de la aplicación y con los que se prevé el funcionamiento corriente. Estos casos de uso irán desde gestionar su perfil hasta gestionar los artículos que desee subir a la plataforma. Además, podrá chatear con los propietarios de los artículos que le interesen además de valorarlos.

- **Cerrar sesión:** A través de un botón de cierre de sesión, el usuario podrá cerrar su sesión en la aplicación y volver a la página principal de la aplicación con el rol de invitado. La vista de cierre de sesión al clicar en cierre de sesión únicamente borrará el objeto persistente de usuario en la petición y le redirigirá a la vista de inicio que mostrará al usuario la plantilla de inicio.

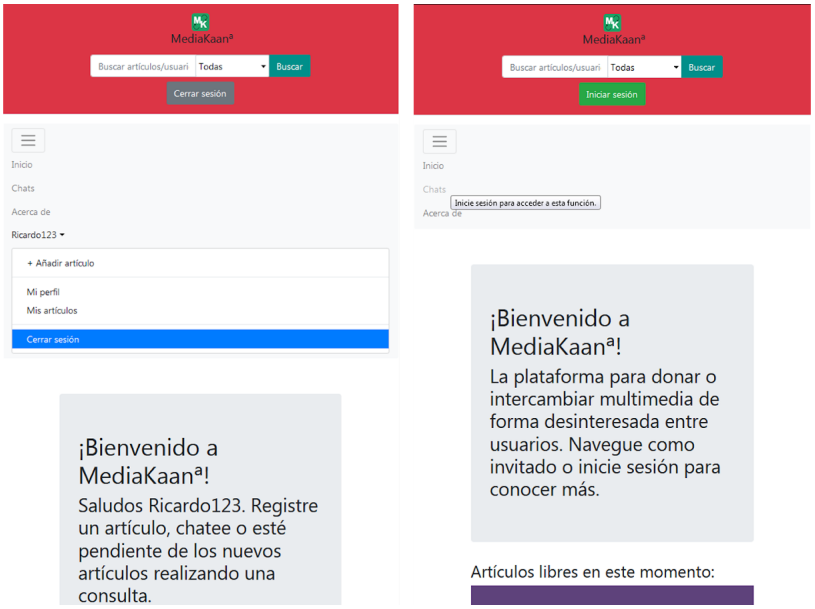


Figura 13. Interfaz de inicio vista desde un móvil antes y después de cerrar sesión

- **Consultar perfiles:** El comportamiento de este caso de uso será el mismo que el del invitado con el añadido de que el usuario podrá consultar la información de su propio perfil. Si el usuario clica en el apartado *Mi perfil* la vista de perfil de usuario a partir de una consulta a la base de datos con el objeto usuario persistente de la petición renderizará la información general del usuario en la plantilla de perfil de usuario donde también podrá editarla desde otro caso de uso.

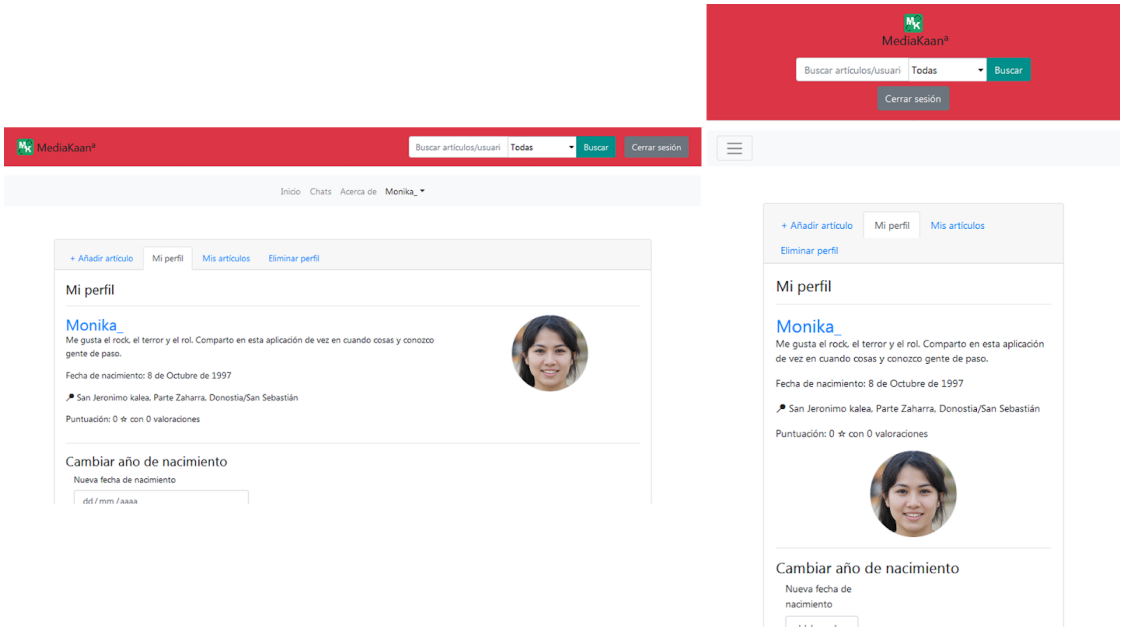


Figura 14. Interfaz de perfil de usuario vista desde un ordenador y desde un móvil

- **Valorar perfiles:** Si el usuario ha recibido un artículo de otro usuario y por tanto este se lo ha asignado, podrá valorar el artículo recibido desde la plantilla *Mis artículos recibidos* con un control de valoración de 5 estrellas. Tras valorar el artículo y enviar dicha valoración, la vista de artículos recibidos comprobará cual de los artículos de la lista de la plantilla ha sido valorado, si está dentro del rango y si dicho artículo identificado por id en la base de datos ha sido asignado al usuario que lo valora (consultado con el objeto usuario persistente de la petición). Tras esas comprobaciones, si la valoración ha sido exitosa se obtendrá dicho objeto artículo de la base de datos y se pondrá la puntuación escogida por el usuario en su campo de valoración actualizando el artículo en la base de datos. Dado que esta valoración también contribuirá a la global del usuario, el siguiente paso que realizará la vista será obtener de la base de datos el propietario del artículo y actualizar sus puntos de valoración con los nuevos dados por el usuario asignado subiendo en uno también el número de valoraciones del perfil. Actualizados los puntos del propietario y del artículo, la vista de artículos recibidos renderizará el artículo valorado sin el control de valoración y con los puntos puestos por el usuario asignado. La valoración global de los usuarios vistas desde algunas plantillas se calcularán en las propias plantillas realizando una división entre el atributo de puntos de valoración del usuario y el atributo de las veces que ha sido valorado.

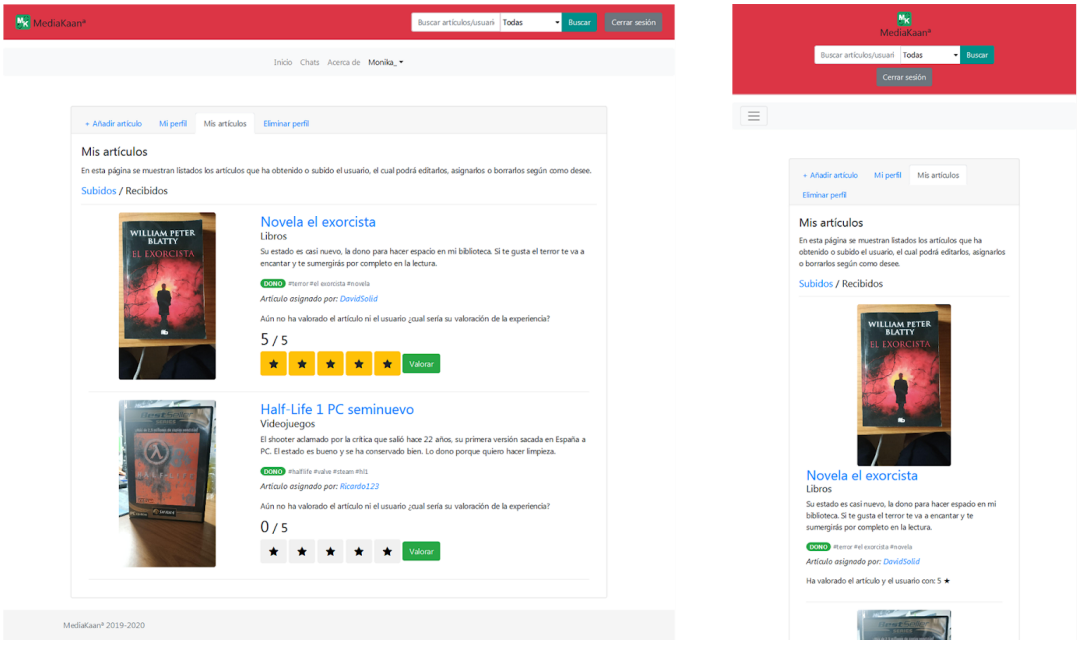


Figura 15. Interfaz de artículos recibidos antes de valorar un artículo (PC) y después de valorarlo (móvil)

- **Editar perfil:** Un usuario podrá editar su información desde la plantilla del perfil de usuario en el apartado *Mi perfil*. La información que podrá cambiar será su contraseña, su biografía, su avatar, su email, su fecha de nacimiento y su ubicación de usuario, sujetos todos estos campos a las mismas validaciones que había en el caso de uso de *Registrarse* del invitado. Cada uno de estos campos tendrá su propio control de cambio con texto de ayuda, de forma que si el usuario cambia uno de sus campos y

envía el cambio, la vista del perfil de usuario detectará que campo se ha actualizado y tras una validación individual actualizará el objeto de información del usuario en la base de datos. Si la actualización de uno de los campos es exitosa, la vista renderizará la plantilla del perfil de usuario con un mensaje de éxito y si por el contrario ha habido errores, la vista renderizará la plantilla del perfil de usuario con un mensaje de error bajo el campo mal rellenado.

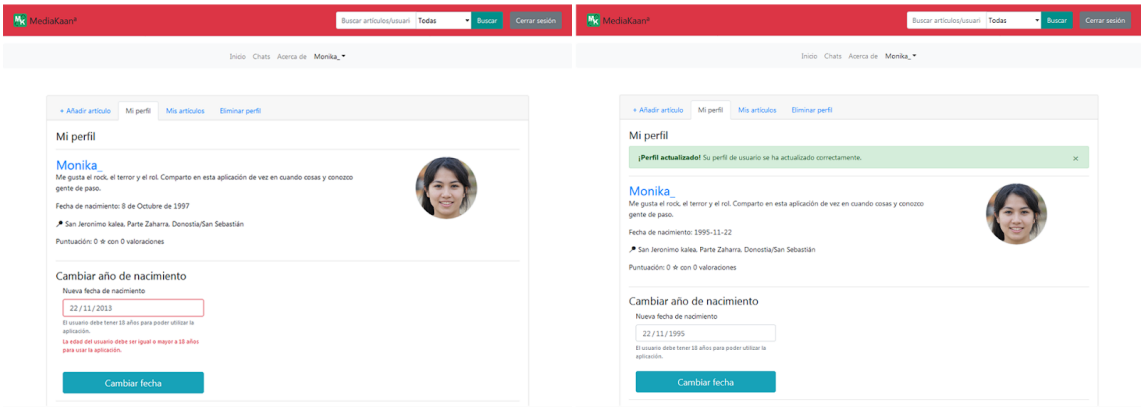


Figura 16. Interfaz de perfil del usuario con cambio en fecha de nacimiento con errores y sin errores

- **Eliminar cuenta:** Si el usuario desea eliminar su cuenta, este podrá hacerlo desde la plantilla de eliminar cuenta donde deberá dar su contraseña para hacerlo. Una vez haga esto y presione el botón de eliminar cuenta, la vista de eliminar cuenta tomará de la base de datos el objeto del usuario correspondiente a partir de la contraseña introducida por el usuario y el nombre de usuario tomado del objeto usuario persistente en la petición. Si el objeto del usuario se ha tomado correctamente, este será borrado de la base de datos junto a los demás objetos del usuario (artículos, conversaciones, datos adicionales de usuario...) y tras borrar el objeto persistente del usuario en la petición, se le redireccionará al ahora invitado a la vista de inicio que renderizará la plantilla índice. En el caso de que la contraseña proporcionada sea incorrecta, la vista cargará nuevamente la plantilla con un mensaje de error.

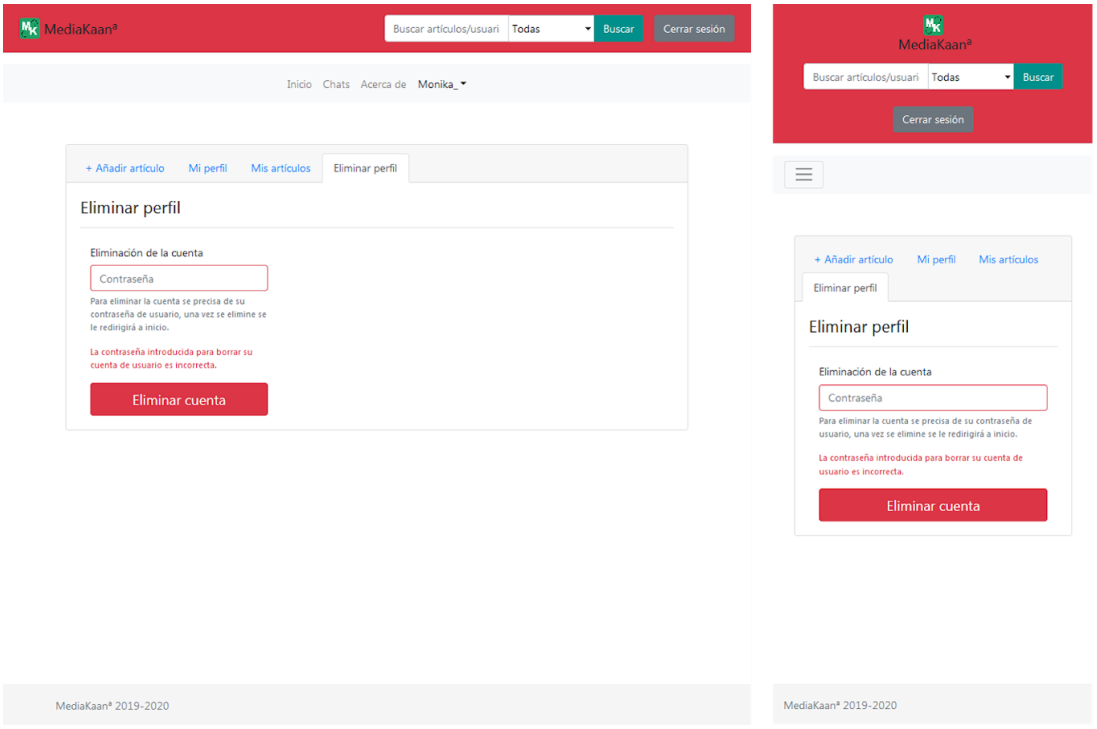


Figura 17. Interfaz de eliminación de cuenta con errores de validación vista desde dos dispositivos

- **Enviar sugerencia:** Actuará de igual forma que el caso de uso del invitado con la particularidad de que la vista si detecta usuario en la petición, colocará en el mensaje enviado por correo electrónico al desarrollador la autoría del emisor.

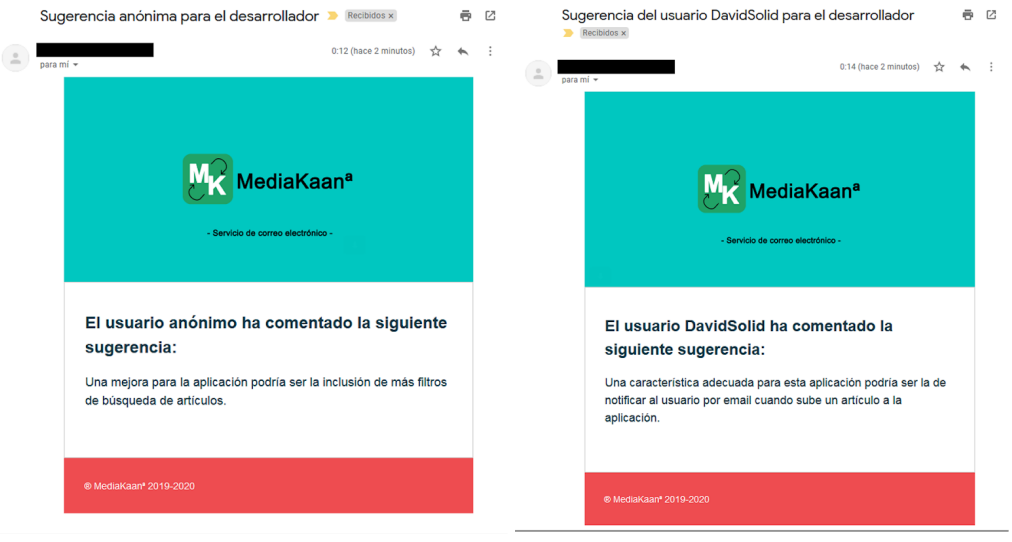


Figura 18. Correo electrónico de sugerencia con usuario y sugerencia con invitado (anónima)

- **Consultar artículos:** El caso de uso tendrá el mismo propósito que el comentado en el rol de invitado con el cambio de que el usuario podrá también consultar sus propios artículos recibidos o subidos desde el apartado *Mis artículos*. La vista de artículos subidos por el usuario tomará de la base de datos con una consulta con el objeto

persistente usuario de la petición sus artículos y mostrará listada toda su información renderizada en la plantilla con controles de borrado de artículos y de asignación de artículos. Si un usuario no dispone de ningún artículo, la plantilla renderizará un mensaje al usuario animándolo a que registre artículos.

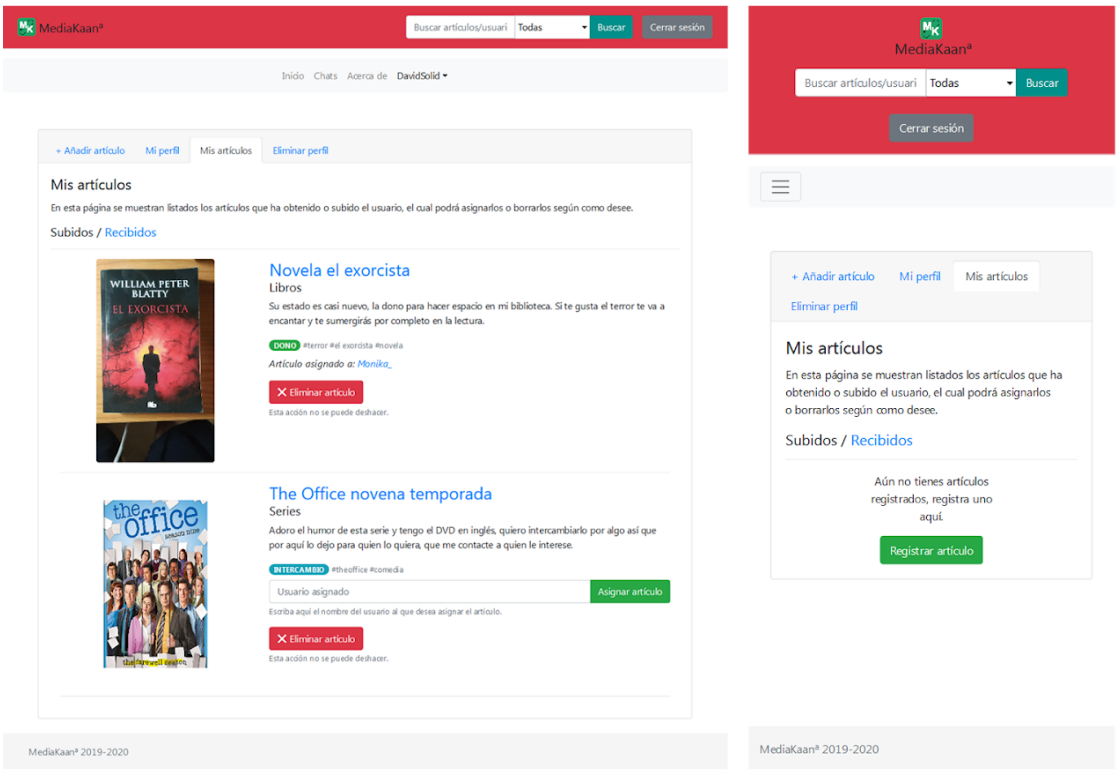


Figura 19. Interfaz de artículos del usuario con artículos y sin ellos

La vista de artículos asignados por otra parte, realizará la misma consulta a la base de datos con el objeto persistente usuario de la petición tomando aquellos en los que ha sido asignado devolviéndolos por la plantilla de artículos recibidos por el usuario y dándole al usuario un control de valoración de 5 estrellas en aquellos que no haya valorado. Si el usuario no ha recibido artículo alguno, la plantilla de artículos recibidos mostrará un mensaje animándolo a buscar algún artículo que le interese.

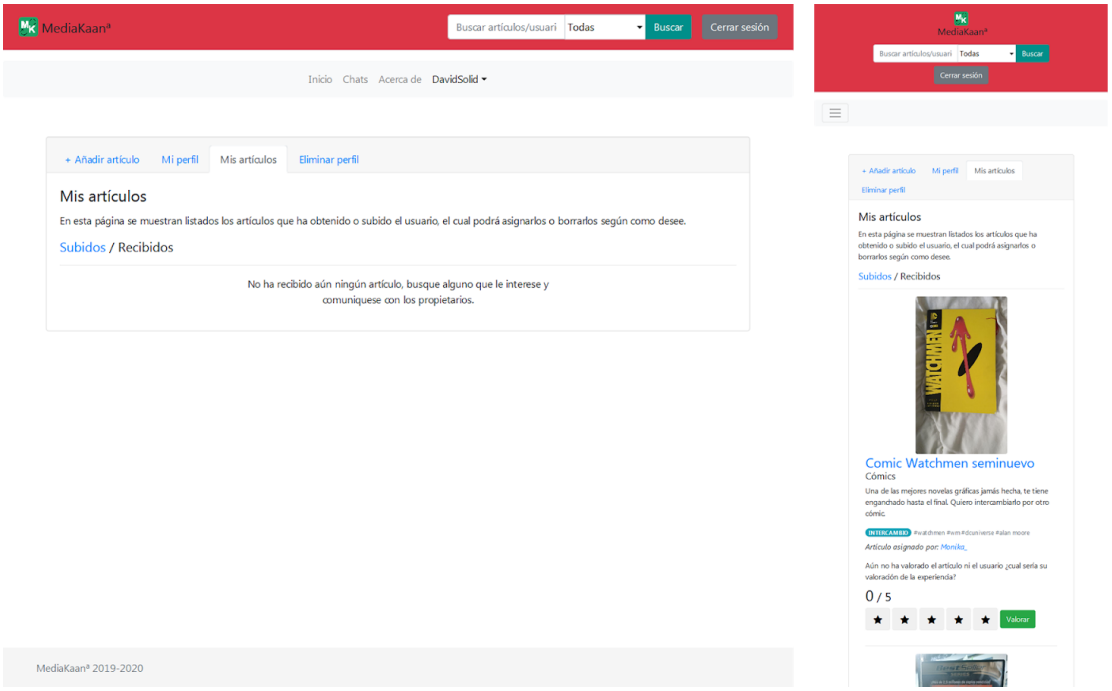


Figura 20. Interfaz de artículos recibidos por el usuario sin artículos y con artículos (vista desde distintos dispositivos)

- **Registrar artículo:** Si el usuario quiere registrar un artículo en la aplicación podrá hacerlo a partir del apartado *Registrar artículo*, que procesará la vista de registrar artículo renderizando la plantilla con los campos de información que el usuario debe rellenar junto a un texto de ayuda debajo de cada uno de ellos. Estos datos comprenderán una foto del artículo, un título, su descripción, la acción de lo que se quiere hacer con él, su categoría y los tags o etiquetas, que serán pequeñas palabras para facilitar búsquedas de ese tipo de artículo. La vista detectará dicha petición una vez el usuario registre el artículo enviando los datos y procederá a comprobar si todos los datos del formulario son correctos. En caso positivo, la vista creará el objeto artículo en la base de datos, los objetos tags del artículo (si ya no estaban en la aplicación) referenciados al mismo, referenciará dentro del objeto de información adicional del usuario el artículo añadido por él y devolverá un mensaje de éxito renderizado en la plantilla de registrar artículo. En caso contrario, la plantilla de registrar artículo se renderizará con los mensajes de error debajo de cada campo del formulario mal rellenado. Este caso se detallará más con un diagrama de secuencia posteriormente.

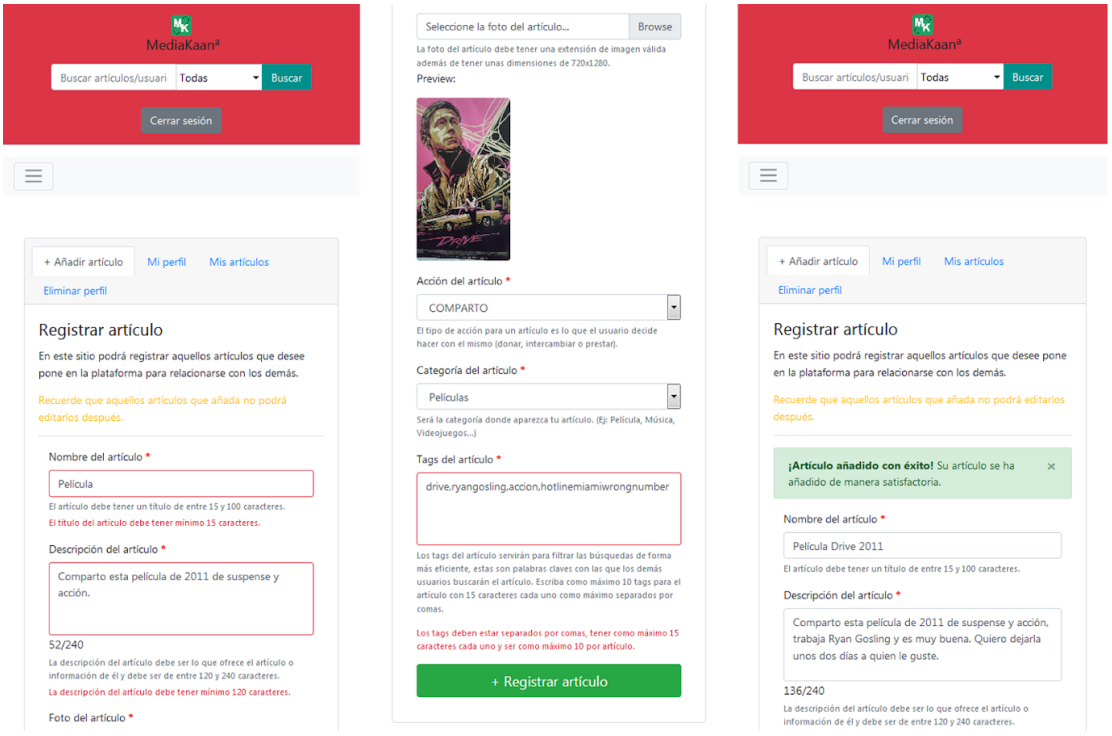


Figura 21. Interfaz de registrar artículo con errores y sin errores

- **Asignar artículo:** Tras registrar un artículo, el usuario podrá asignarlo a cualquier otro usuario que haya o no haya establecido comunicación. Para asignarlo, el usuario escribirá en el campo de asignación de uno de sus artículos listados en la plantilla el nombre del usuario al que desea asignar el artículo. Enviado ese dato, la vista de artículos del usuario comprobará que artículo se ha asignado y si el campo de asignación es correcto, después de ello comprobará a partir de una consulta a la base de datos si el nombre de usuario corresponde a algún usuario y si no es el nombre del mismo usuario. Si todo ello es correcto y el artículo es del usuario que asigna, se obtendrá dicho artículo de la base de datos y se actualizará su campo de asignación con el objeto usuario asignado tomado a partir del nombre. Finalmente, el objeto de datos adicionales del usuario asignado se actualizará referenciando el artículo que ha recibido y se le informará al usuario que asigna renderizando la plantilla con un mensaje de éxito haciendo referencia al nombre de a quien ha sido asignado. Si por el contrario, el nombre del usuario no consta en los registros, la plantilla será renderizada con un mensaje de error.

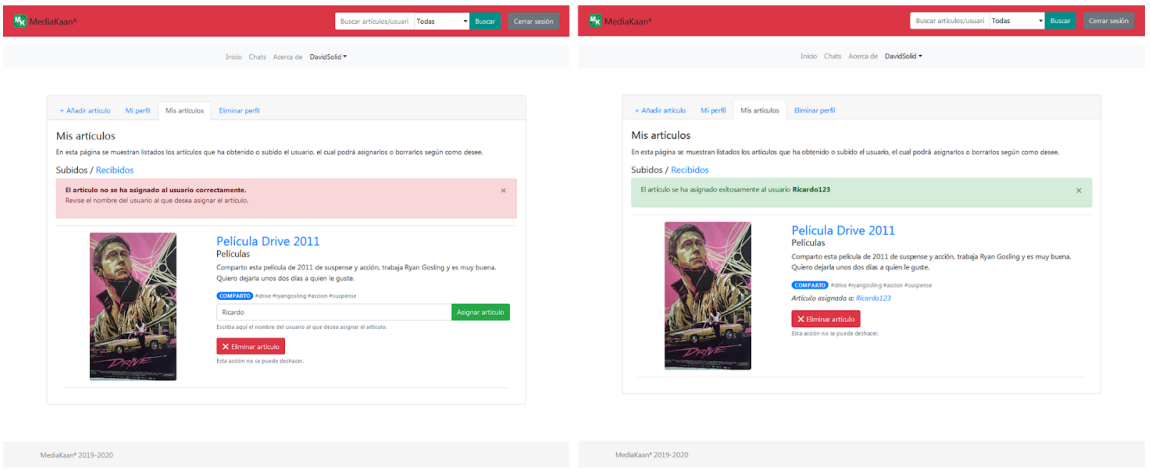


Figura 22. Interfaz de artículos del usuario con error de asignación y con asignación correcta

- **Eliminar artículo:** Tras haber registrado un artículo, si el usuario desea eliminarlo podrá hacerlo desde la plantilla de artículos del usuario clicando en el botón eliminar de uno de sus artículos listados. La vista de artículos del usuario una vez reciba la petición de borrado con el identificador del artículo comprobará en la base de datos si existe y si su propietario es el objeto usuario persistente de la petición. En caso de que sea así, la vista tomará el artículo y borrará la referencia del artículo en el objeto de información adicional del usuario y si este además estaba asignado, borrará la referencia del artículo en el objeto de información adicional del usuario asignado. Finalmente, el objeto del artículo será borrado y tomando su título se informará por un mensaje en la plantilla de artículos del usuario de que el artículo con dicho título ha sido eliminado.

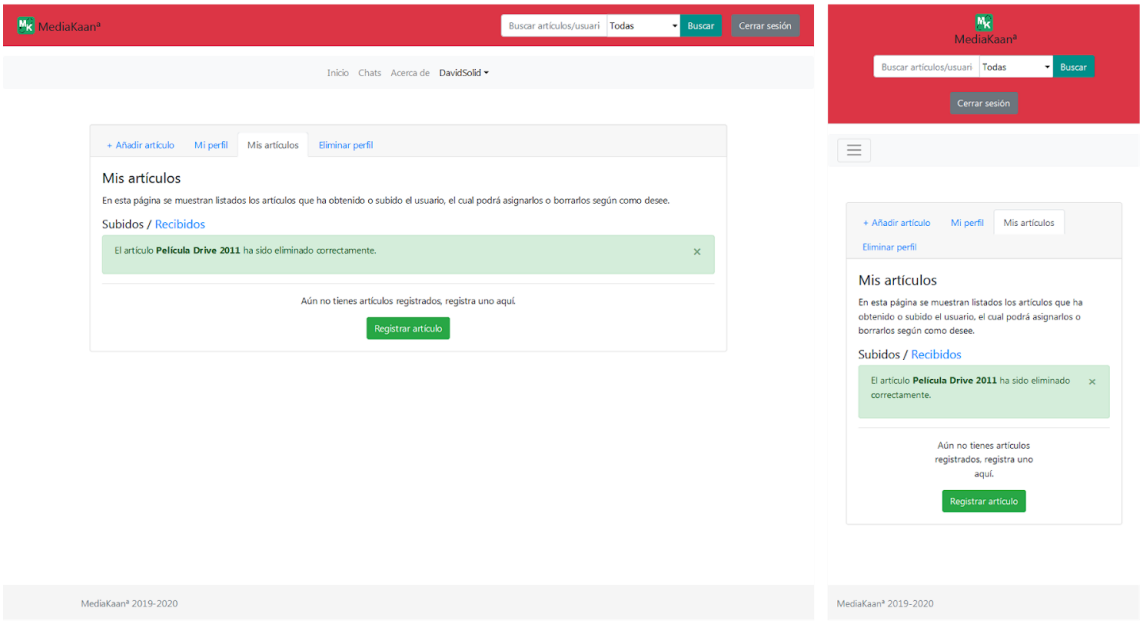


Figura 23. Interfaz de artículos del usuario con mensaje de borrado exitoso (vista desde distintos dispositivos)

- **Enviar mensaje:** Este caso de uso será el que permita la comunicación entre el propietario de un artículo y el interesado en conseguirlo, cada artículo tendrá un botón para que el interesado abra conversación con el propietario desde la plantilla de artículo individual, este botón en esta plantilla solo estará habilitado para los usuarios que hayan iniciado sesión y esté interesado por el artículo. Cada botón en esta plantilla tendrá una dirección url hacia la vista de enviar mensaje junto a la id del artículo por el que tiene interés el usuario. Cuando el usuario haga click en dicho botón de comunicación, la vista renderizará la plantilla de enviar mensaje con la conversación activa sobre el artículo con el propietario junto a las demás conversaciones con propietarios e interesados que ya tenía abiertas. La vista para hacer esto primeramente cargará de la base de datos las conversaciones en las que el usuario es el interesado y en las que el usuario es el propietario del artículo, seguido de ello comprobará si la id del artículo corresponde a alguno en la base de datos y si es así tomará dicho artículo. Después, la vista comprobará si ya existe una conversación entre el usuario propietario e interesado en la base de datos y la creará o obtendrá dependiendo del caso distinguiendo si el usuario es propietario o interesado.

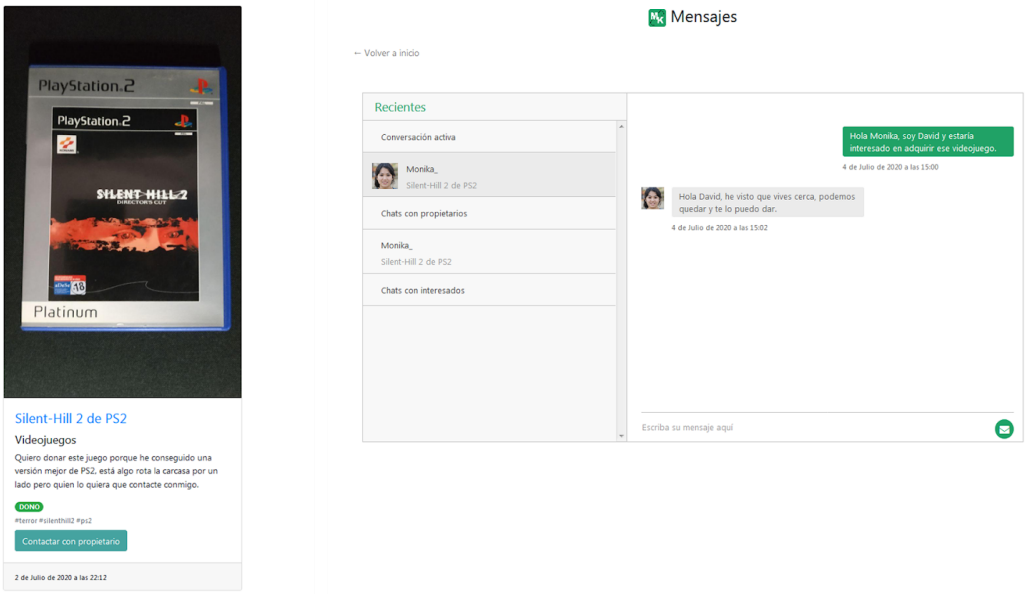


Figura 24. Interfaz de artículo individual con botón de comunicación habilitado e interfaz de enviar mensaje con una conversación abierta con la propietaria del artículo

En este punto, si el usuario interesado envía un mensaje la vista de enviar mensaje detectará la petición de envío y validando el texto enviado y que en la conversación el artículo no ha sido asignado, guardará dicho mensaje en la base de datos referenciado quien lo envía y quien lo recibe y referenciando también a que conversación pertenece. Una vez el envío sea correcto, la plantilla de envío de mensaje será cargada de nuevo con la conversación activa sobre el artículo con el nuevo mensaje y el propietario en su apartado de conversaciones verá que tiene una nueva conversación. Esta plantilla de envío de mensajes también renderizará los demás enlaces de las conversaciones abiertas con propietarios e interesados para hacer dicha conversación activa si se clica en ella.

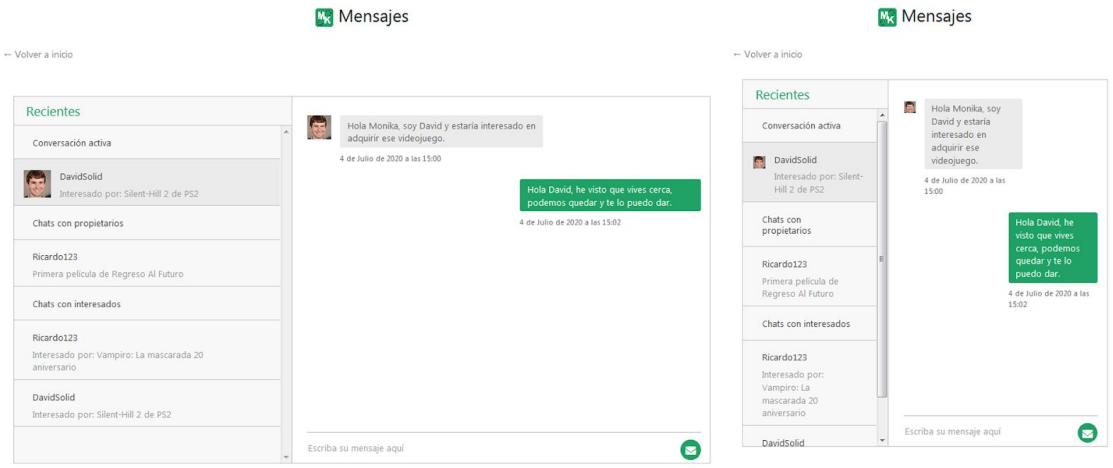


Figura 25. Interfaz de enviar mensaje desde distintos dispositivos desde el punto de vista del propietario

4.1.3. Rol de administrador

El rol de administrador servirá para administrar ciertos conjuntos de datos de la aplicación además de servir para realizar distintas pruebas durante el desarrollo. Estos datos que se podrán manipular serán los relativos a usuarios y artículos dentro de la aplicación. Los casos de uso del administrador no serán estrictamente iguales a los vistos en otros roles ya que se usará la interfaz de administrador de Django.

Los casos de uso de gestión del administrador como son **añadir usuarios/artículos**, **editar usuarios/artículos**, **consultar usuarios/artículos** y **eliminar usuarios/artículos** tendrán la misma forma de manejo desde el panel del administrador como con cualquier otro modelo de datos.

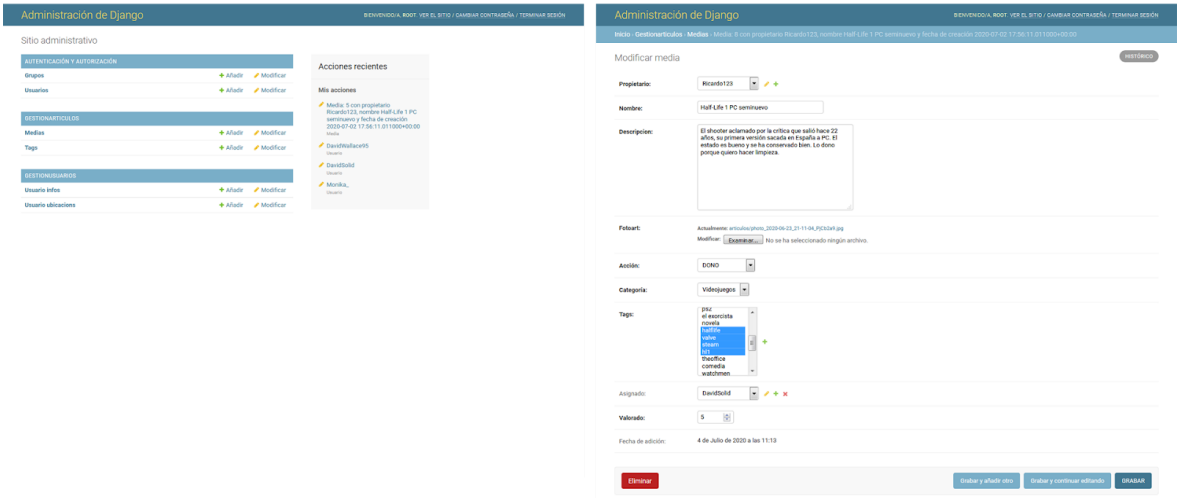


Figura 26. Panel de administración de Django de la aplicación y actualización de un artículo

Para añadir estos datos se hará a través de un formulario con validaciones, se eliminarán seleccionando aquellas filas de la tabla que se deseen y se editarán seleccionando la fila de la tabla que se quiera editar en un formulario con validaciones.

El cambio significativo estará entre los casos de consultas de datos, que entre artículos y usuarios vendrán mejores detallados en cuanto a filtros debido a los datos que se manejen (filtros de fechas, filtros de categoría/acción/tags en artículo...) pero que funcionarán principalmente escribiendo lo que se desee buscar en un campo de búsqueda. En todos ellos se hacen consultas, actualizaciones, inserciones y eliminaciones en la base de datos dependiendo de lo que se haga con los objetos.

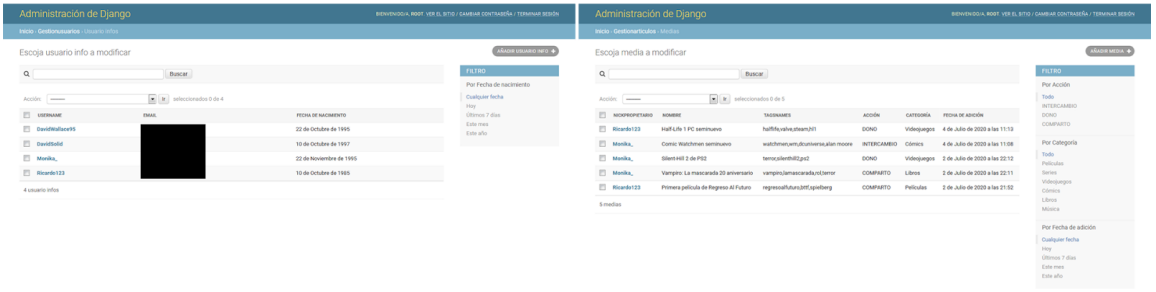


Figura 27. Panel de consulta del administrador del modelo de datos adicionales del usuario y del modelo de artículos

4.2. Modelo de dominio

El modelo de dominio correspondiente al análisis de requisitos que hemos diseñado se muestra en la figura 28 utilizando la notación UML. Dentro de este nos encontraríamos con siete clases: Usuario, UsuarioInfo, Ubicación, Media, Tag, Chat y Mensaje.

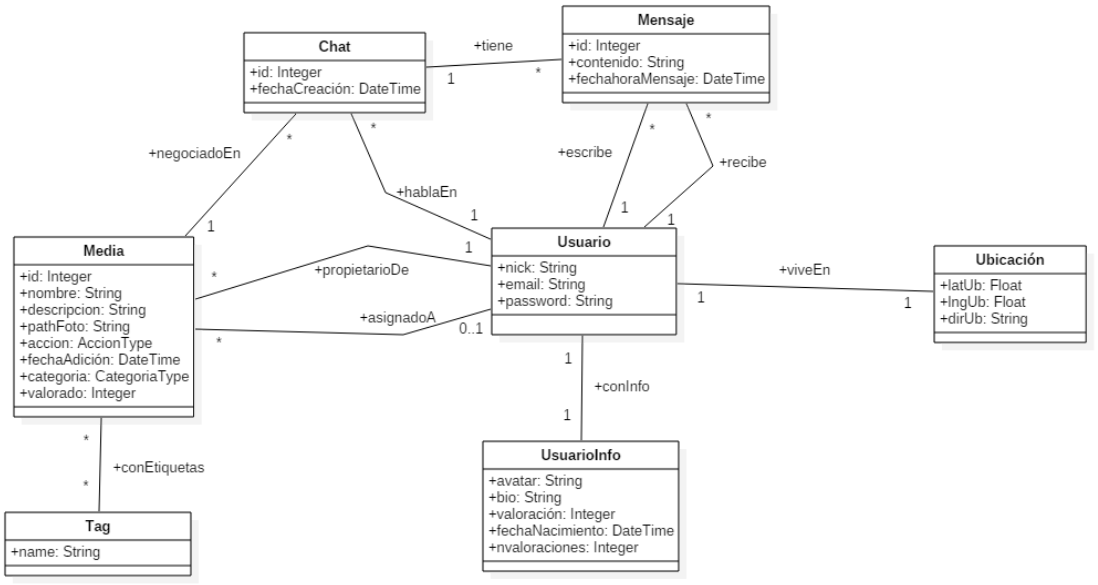


Figura 28. Modelo de dominio

Este modelo de dominio corresponde al de una base de datos no relacional y está formado por las siguientes siete clases con sus respectivos atributos:

- **Usuario:** Almacenará los datos identificativos de los usuarios. Es la clase utilizada por todo proyecto de Django perteneciente a su sistema de autentificación.

  Los atributos de la clase son los siguientes:

  - **nick (clave primaria-string):** Será el nombre de cada usuario y se utilizará como el identificador principal del usuario dentro de la aplicación.
  - **email (string):** El email del usuario, cada usuario dispondrá de uno diferente y servirá para establecer una vía de comunicación externa con el usuario además de como otro identificador.
  - **pass (string):** Será la clave que el usuario elija para identificarse cuando inicie sesión en la aplicación.

- **UsuarioInfo:** Comprenderá la información adicional de los usuarios que servirá a otros usuarios para conocer ciertas características sociales de otros a la hora de decidir si comunicarse o no con ellos.

  Los atributos de la clase son los siguientes:

  - **usuario (clave primaria extranjera-int):** El identificador del usuario al que pertenece la información de este documento.
  - **avatar (string):** Será la imagen que mostrará el usuario a los demás en su perfil, la cual quedará localizada en este campo por medio de la ruta de la imagen.
  - **bio (string)**: Una descripción breve del usuario sobre él que podrán visualizar los demás usuarios, reunirá de manera general lo que el usuario quiera que se conozca de él.
  - **valoración (integer):** El número entero de puntos que dispondrá el usuario conforme a las valoraciones que los demás usuarios realicen de él.
  - **n_valoraciones (integer):** El número de usuarios que hayan valorado los artículos asignados por el usuario.
  - **fechaNacimiento (datetime):** Corresponderá a la fecha de nacimiento del usuario.
  - **ubicacion (objeto):** Guardará un objeto con la información del documento de la ubicación del usuario.
  - **articulos (array-int):** Guardará en un vector aquellos identificadores de los artículos subidos por el usuario.
  - **articulos_rec (array-int):** Guardará en un vector aquellos identificadores de los artículos recibidos o asignados para el usuario.

- **Ubicación:** Reunirá la información de la ubicación de los usuarios que se le mostrará a los demás usuarios para que conozcan donde el usuario tiene los artículos que ha puesto en disposición dentro de la plataforma.

Los atributos de la clase son los siguientes:

- **usuario (clave primaria extranjera-int):** El identificador del usuario al que pertenece la información de este documento.
- **latUb (float):** Latitud de la ubicación del usuario, representada con un decimal.
- **lngUb (float):** Longitud de la ubicación del usuario, representada con un decimal.
- **dirUb (string):** Dirección del usuario calculada por la latitud y longitud de la ubicación que ha decidido el usuario.

- **Media:** Registrará la información que tendrán los artículos dentro de la aplicación que podrá ser consultada por los demás usuarios.

  Los atributos de la clase son los siguientes:

  - **media_id (clave primaria-int):** Identificador único entero de cada artículo de la plataforma.
  - **propietario (clave extranjera-int):** Identificador del usuario propietario del artículo.
  - **nombre (string):** El título que tendrá el artículo que se haya puesto en la plataforma, será una información muy breve de lo que es.
  - **descripción (string):** La descripción general del artículo en la plataforma que acompañará al título del artículo dando información adicional como puede ser el estado del artículo, que año se consiguió, que edición, etc.
  - **fotoart (string):** Será la imagen que se mostrará a los usuarios del artículo, estará localizada en este campo por medio de su ruta de archivo.
  - **acción (enum-string):** Será una palabra que reúna la acción que el usuario quiere realizar con el artículo que podrá ser compartir, donar o intercambiar. (COMPARTO, INTERCAMBIO, DONO)
  - **categoría (enum-string):** La categoría a la que corresponderá el artículo que podrá estar dentro de: películas, videojuegos, cómics, música, libros o series.
  - **tags (array-int):** Vector con los identificadores de los tags que tendrá el artículo.
  - **asignado (clave extranjera-int):** Identificador del usuario que ha recibido el artículo.
  - **valorado (int):** Puntos de 1-5 que ha puesto al artículo el usuario que lo ha recibido.
  - **fechaAdición (datetime):** La fecha que se registrará una vez el artículo se haya añadido a la aplicación por el usuario.

- **Tag:** Complementará la información del documento Media, reunirá información de las etiquetas correspondientes a los artículos con los que poder agilizar la consulta de ellos.

Los atributos de la clase son los siguientes:

- **id (clave primaria-int):** El identificador del tag.
- **name (string):** Una palabra o pequeño conjunto de varias que otorgará información básica del artículo con la que los demás usuarios podrán hacer búsquedas en la aplicación.

- **Chat:** Tendrá información sobre las sesiones de conversaciones que se creen de cada artículo de aquellos usuarios que estén interesados en adquirirlos con el propietario.

  Los atributos de la clase son los siguientes:

  - **chat_id (clave primaria-int):** Identificador único entero de la sesión del chat.
  - **articulo (clave extranjera-int):** Identificador del artículo que le interesa al usuario interesado.
  - **interesado (clave extranjera-int):** Identificador del usuario interesado por un artículo.
  - **fechaCreación (datetime):** Fecha de la creación chat que se registrará cuando un usuario interesado decida entablar conversación con el propietario del artículo.
  - **msgs (array-int):** Vector de identificadores de los mensajes compartidos entre el interesado por el artículo y su propietario.

- **Mensaje:** Almacenará los datos de los mensajes de las conversaciones que se creen de los usuarios interesados de cada artículo. Estos mensajes podrán ser escritos por propietario o por el interesado en su comunicación.

  Los atributos de la clase son los siguientes:

  - **msg_id (clave primaria-int):** Identificador único entero del mensaje.
  - **chat (clave extranjera-int):** Identificador de la conversación a la que pertenece el mensaje.
  - **contenido (string):** Contenido escrito del mensaje por el interesado o por el propietario del artículo.
  - **escribe (clave extranjera-int):** Identificador del usuario que escribe el mensaje (propietario o interesado).
  - **recibe (clave extranjera-int):** Identificador del usuario que recibe el mensaje (propietario o interesado).
  - **fechaHoraMsg (datetime):** La fecha y hora en la que se ha enviado el mensaje del interesado al propietario o viceversa.

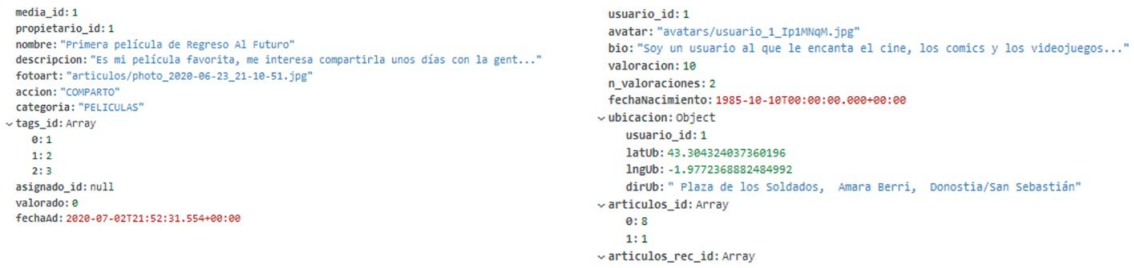


Figura 29. Documento de ejemplo de Media y UsuarioInfo almacenado en la base de datos

4.3. Diseño mediante Diagramas de Secuencia

Un diagrama de secuencia es una gran herramienta para establecer los métodos de cada clase y el paso de parámetros entre las capas de la aplicación durante un caso de uso ejecutado por un actor. Uno de los casos que en esta aplicación será uno de los más ejecutados para su propósito principal será el caso del rol usuario *Registrar artículo*, el cual fue introducido brevemente en una sección anterior de este capítulo.
El diagrama de secuencia correspondiente a este caso de uso se puede apreciar en la siguiente figura y **sigue la arquitectura Model Template View** de DJango con la que vienen implementados todos los casos de uso, una variante de la conocida arquitectura Modelo Vista Controlador.

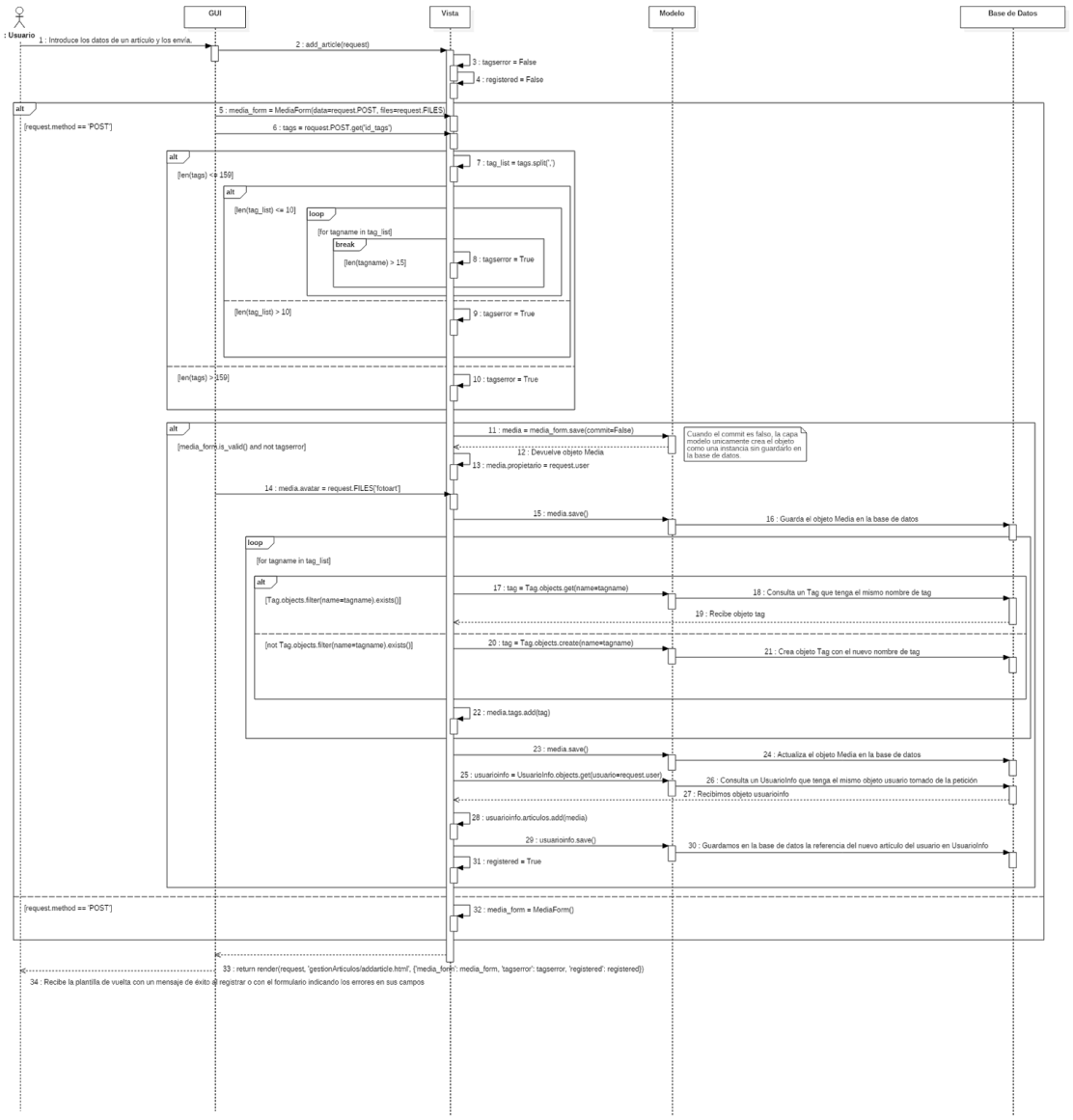


Figura 30. Diagrama de flujo del caso de uso Registrar artículo

Puesto que con este diagrama se ve la comunicación entre capas general que tienen los casos de uso con la arquitectura ofrecida por Django y debido a las limitaciones de

extensión de la memoria, se ha decidido no colocar los de cada caso de uso. Adicionalmente, si se desea visualizar otro diagrama de otro caso de uso, este puede ser consultado en el **Anexo E** concretando el intercambio de datos entre capas del caso de uso *Iniciar Sesión*.

5. Tecnologías

Durante el siguiente capítulo describiremos lo que nosotros hemos dividido cómo dos categorías de tecnologías, las **principales** y las **específicas**. Para las principales se citarán aquellas elegidas cuyo uso ha sido general en el desarrollo de las funcionalidades de la aplicación, mientras que para las específicas se citarán las escogidas para soluciones más particulares o excepcionales.

A continuación se hará una breve introducción de cada una de ellas, el porque se han escogido y por último las ventajas que ellas nos presentan.

5.1. Tecnologías principales

5.1.1. Django

Django es un framework de desarrollo web de código abierto escrito en Python que ha recibido soporte y actualizaciones desde que se lanzó su primera versión en Septiembre de 2008, lo que ha llevado a que sea uno de los más populares según la información ofrecida por **HotFrameworks**[2]. Tal es su popularidad que forma parte de sitios web con gran reconocimiento como *Instagram, Bitbucket, The Washington Times* y *Mozilla*.

La razón principal por la que se escogió dicha tecnología para desarrollar la aplicación de la que es objeto este proyecto fue porque se conoció de forma breve en una asignatura de la especialidad, lo que creó atención en mí e hizo que investigará sobre dicho framework. Django me ofrecía muchas ventajas como integrar en aplicaciones el patrón MVC que tanto se había estudiado durante el grado para las aplicaciones e integrar la filosofía de baterías incluídas[3], lo que hacía posible que capas de la aplicación como la **seguridad**, **internacionalización** o otras características más vinieran de serie sin que el desarrollador tuviera que molestarse en integrarlas.

Un ejemplo de ello sería el sistema de usuarios que trae, donde el propio Django administra la forma de gestionar las cookies sin guardar datos de usuario en ellas y almacenando una clave con la que acceder a los datos en la base de datos de los usuarios con sesión iniciada. Además de guardar las contraseñas de los usuarios ya hasheadas al crear un usuario en su sistema.

Adicionalmente, Django posee una API robusta para la gestión de sistemas de bases de datos, permitiendo que se ejecuten operaciones de consulta o inserción de manera ágil y asegurándose de lo que se registra en ella a través de un sistema de sanitización ya integrado evitando inyecciones de código o de instrucciones para la base de datos.

[2] Web de ranking de frameworks de desarrollo web populares https://hotframeworks.com/

[3] La filosofía de baterías incluídas es aquella en la que se da al usuario una biblioteca estándar rica y versátil en funcionalidades para que no se preocupe por desarrollar o descargar paquetes por separado.

5.1.2. MongoDB

MongoDB es un sistema de bases de datos no relacional de código abierto orientado a documentos lanzado sobre 2009. Actualmente es uno de los sistemas más utilizados por las aplicaciones modernas soportadas en la nube, principalmente por la estructura de datos que maneja bajo el principio clave-valor permitiendo la escalabilidad de grandes volúmenes de datos y el acceso rápido de los mismos debido a su diseño.
Actualmente ocupa el puesto número cinco dentro de los sistemas de bases de datos más utilizados según la información ofrecida por **DB-Engines**[4] y da servicio a organizaciones de renombre como *Google*, *Telefónica*, *Cisco* y *Nokia*.

Esta es una de las tecnologías que también aunque de forma más breve a la anterior se conoce en la especialidad a través de una asignatura y en la que se quería profundizar debido a que las estructuras de datos pseudo JSON (BSON en este caso) que utiliza hacía que las consultas u otras operaciones fueran muy fácil de realizar, además de ser una tecnología idónea para el tipo de aplicación que teníamos que desarrollar. Según la notación de este sistema, estos **documentos** serían las tuplas de una entidad y cada conjunto de ellos para las distintas entidades serían las llamadas **colecciones**. MongoDB además ofrece servicio en la nube gratuito para mantener una base de datos, lo que hacía que de cara a realizar futuras pruebas de la aplicación en la nube fuera una gran ventaja para no mantenerla desde el mismo servicio de hosting decidido.

Por último, tendríamos su inclusión dentro de Django que sería relativamente fácil de implementar gracias al conector **djongo**. Permitiéndonos utilizar la API robusta para las bases de datos que nos ofrece al ser un transpilador de instrucciones SQL a MongoDB, manteniendo así todas las características que nos ofrece Django con los modelos.

5.1.3. JQuery

Sobre JQuery podríamos decir tras haberla conocido de forma extensa en la especialidad que es una librería indispensable para el desarrollo en JavaScript siendo su librería más utilizada y sobre la que se apoyan la mayoría de tecnologías integradas con JS. Según **BuiltWith**[5], desde 2019 esta tecnología se utiliza en el 79,47% del millón de webs más importantes y en el 86% de todos los sitios web.

JQuery nos ofrece características como la selección o modificación ágil por referencia de distintos elementos del DOM, la gestión de eventos, la manipulación de hojas de estilo CSS, etc. Además da soporte a la tecnología Bootstrap de la que hablaremos a continuación. Es por ello que su elección fue clara para el desarrollo del proyecto.

5.1.4. Bootstrap

Bootstrap fue una de las tecnologías que durante el grado fue meramente mencionada como recomendación y a la que no se le llegó a dar un uso real. Esta tecnología es un conjunto de librerías de CSS y JS de código abierto que otorga herramientas de diseño

[4] Web de ranking de sistemas de bases de datos más populares https://db-engines.com/en/ranking
[5] BuiltWith es una empresa especializada en el análisis y seguimiento de las tecnologías web utilizadas para desarrolladores.

para sitios web. Actualmente es el segundo proyecto más destacado de GitHub y es la librería front-end más utilizada.

Contiene gran cantidad de elementos de diseño estandarizados como tipografías, formularios, componentes, botones, iconos, etc. Y sirve a muchos equipos de desarrolladores como una base para establecer su propia librería frontend detallada para lo que desean realizar. Además, uno de sus grandes puntos fuertes es que a partir de su segunda versión soporta el diseño responsive de las aplicaciones por lo que para nuestra aplicación multiplataforma como herramienta de diseño es la tecnología idónea.

5.2. Tecnologías específicas

5.2.1. Leaflet

Leaflet es una librería de código abierto de JavaScript para mapas interactivos.
Dentro de la aplicación nos servirá para dotar a los usuarios con controles para registrar su ubicación y consultar la ubicación de los demás usuarios.

5.2.2. GeoPy

GeoPy es una librería de código abierto para Python para localizar coordenadas de direcciones, ciudades y países a través de diferentes geocodificadores u otras fuentes de datos. Dentro de la aplicación nos ayudará a comprobar y almacenar la dirección de un usuario a partir de los datos que nos otorgue por medio de una plantilla que nos envíe a una vista.

5.2.3. Widget Tweaks

Widget Tweaks es una tecnología asociada a Django (plugin) que sirve para renderizar cada componente de los formularios del mismo en las plantillas de la aplicación pudiendo alterar sus atributos CSS o HTML desde allí.

Esto hace que sea una tecnología esencial en nuestro caso puesto que tendremos que gestionar muchos controles de formularios adecuadamente. Django sin este plugin únicamente será capaz de renderizar los formularios en formato párrafo y en formato de tabla, lo que dificultará la muestra de información de ayuda o de errores asociada a cada componente para el usuario y hará difícil crear estructuras de formularios más complejas.

5.2.4. MathFilters

MathFilters es una tecnología asociada a Django para realizar operaciones matemáticas básicas desde las plantillas sobre datos numéricos de tipo entero o decimal. Django inicialmente proviene únicamente de la suma en enteros para las plantillas por lo que esta herramienta nos ayudará para mostrar aquellos datos que necesiten una operación algo más compleja y así hacer que no recaiga más trabajo dentro de las vistas para ciertos casos.

6. Implementación

La implementación de la aplicación responde a una arquitectura conocida en la ingeniería del software como Modelo-Vista-Controlador. Este tipo de arquitectura es en la que se soportan todos los proyectos creados a través del framework Django con su propia terminología.

Este capítulo servirá para comprender el papel de las capas de dicha arquitectura en la implementación del proyecto con Django, así como los aspectos generales y no tan generales de este framework junto con las tecnologías adicionales utilizadas.

6.1. Estructura del proyecto

Aunque la arquitectura sea uno de los pilares importantes en lo que a software se refiere la estructura de un proyecto es algo a tener en cuenta puesto que gran cantidad de ficheros son utilizados durante el desarrollo y es importante que estén bien organizados y tengan una independencia según el propósito de cada uno. Esto es algo que tiene en cuenta Django y es algo en lo que se hace mucho hincapié en su documentación.

```
.
|-- LICENSE
|-- MediaKaan
|   |-- __init__.py
|   |-- __pycache__
|   |-- asgi.py
|   |-- settings.py
|   |-- urls.py
|   `-- wsgi.py
|-- README.md
|-- gestionArticulos
|   |-- __init__.py
|   |-- __pycache__
|   |-- admin.py
|   |-- apps.py
|   |-- enums.py
|   |-- forms.py
|   |-- migrations
|   |-- models.py
|   |-- tests.py
|   |-- urls.py
|   `-- views.py
|-- gestionMensajeria
|   |-- __init__.py
```

```
|   |-- __pycache__
|   |-- admin.py
|   |-- apps.py
|   |-- migrations
|   |-- models.py
|   |-- tests.py
|   |-- urls.py
|   `-- views.py
|-- gestionUsuarios
|   |-- __init__.py
|   |-- __pycache__
|   |-- admin.py
|   |-- apps.py
|   |-- emails.py
|   |-- forms.py
|   |-- migrations
|   |-- models.py
|   |-- tests.py
|   |-- tokens.py
|   |-- urls.py
|   `-- views.py
|-- manage.py
|-- media
|   |-- articulos
|   `-- avatars
|-- requeriments.txt
|-- static
|   |-- gestionArticulos
|   |   |-- css
|   |   |-- img
|   |   `-- js
|   |        |-- componentsutils.js
|   |        `-- starrating.js
|   |-- gestionMensajeria
|   |   |-- css
|   |   |   `-- messages.css
|   |   |-- img
|   |   `-- js
|   `-- gestionUsuarios
|      |-- css
|      |   |-- base.css
|      |   |-- bootstrap.css
|      |   `-- signin.css
|      |-- img
|      |   |-- articulo.jpg
|      |   |-- author.jpg
|      |   `-- logo
|      |      |-- books.png
|      |      |-- comics.png
```

```
|      |      |-- email-header.png
|      |      |-- mediak-logo.ico
|      |      |-- mediak-logo.png
|      |      |-- motivation.png
|      |      |-- movies.png
|      |      |-- music.png
|      |      |-- series.png
|      |      |-- trade-scene.png
|      |      `-- vgames.png
|      `-- js
|             |-- bootstrap.js
|             |-- jquery-3.4.1.min.js
|             |-- popper.js
|             |-- registerubication.js
|             |-- showubication.js
|             `-- signupcounter.js
`-- templates
    |-- about.html
    |-- gestionArticulos
    |   |-- addarticle.html
    |   |-- article.html
    |   |-- articles.html
    |   |-- myarticles.html
    |   `-- recarticles.html
    |-- gestionMensajeria
    |   `-- messages.html
    |-- gestionUsuarios
    |   |-- activateacc.html
    |   |-- base.html
    |   |-- changepass.html
    |   |-- deleteprofile.html
    |   |-- email
    |   |   |-- email_active.html
    |   |   |-- email_recovery.html
    |   |   `-- email_suggestion.html
    |   |-- forgetpass.html
    |   |-- registerubication.html
    |   |-- signin.html
    |   |-- signup.html
    |   |-- user.html
    |   |-- userprofile.html
    |   `-- users.html
    `-- index.html
36 directories, 140 files
```

Un proyecto de Django inicialmente poseé una carpeta principal con el nombre del proyecto, en nuestro caso MediaKaan. Esta primera carpeta es la que posee los

ficheros de configuración globales del proyecto además de un fichero con las urls registradas que tendrá nuestra aplicación.

Entre los ficheros de configuración que tiene un proyecto dentro de esta carpeta el más importante es el fichero **settings.py**. Este fichero será entre otras cosas el encargado de que queden referenciados el directorio base y los directorios adicionales que se van a utilizar desde las diferentes partes, las aplicaciones instaladas o widgets del proyecto utilizadas, la configuración de la conexión de la base de datos que utilizaremos y el huso horario del servidor o el idioma que tendrá el proyecto, entre otro tipo de cosas.

Un proyecto en Django debe tener dentro de los directorios adicionales en la configuración los siguientes tres: **templates**, **static** y **media**. El directorio **templates** es aquel en el que residirán las plantillas web (.html) que formarán la interfaz gráfica, el **static** en el que estarán ubicados aquellos ficheros con imágenes de la aplicación, hojas de estilo para las plantillas (.css) o librerías del lado cliente (.js) y el **media** donde se guardarán generalmente aquellas imágenes o otros recursos que los usuarios suban a la aplicación. Tanto **templates**, **static** y **media** tienen una estructura más dividida internamente concorde a las aplicaciones que formen el proyecto y los recursos correspondientes que estas usen, aunque esto se comentará más a fondo durante este capítulo.

```
# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Directorios adicionales
TEMPLATE_DIR = os.path.join(BASE_DIR,'templates')
STATIC_DIR = os.path.join(BASE_DIR,'static')
MEDIA_DIR = os.path.join(BASE_DIR,'media')
```

Por último dentro de esta carpeta principal tendríamos el fichero principal de urls del proyecto **urls.py**, desde donde se gestionan los caminos a las vistas de cada aplicación o en general como puede ser la página de inicio, la de deslogueo del usuario y los sitios del administrador.

```
urlpatterns = [
      path('admin/', admin.site.urls),
      url(r'^$', views.index, name='index'),
      url(r'^about/$', views.about, name='about'),
      url(r'^gestionUsuarios/', include('gestionUsuarios.urls')),
      url(r'^gestionArticulos/', include('gestionArticulos.urls')),
       url(r'^gestionMensajeria/', include('gestionMensajeria.urls')),
      url(r'^logout/$', views.user_logout, name='logout'),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

6.2. Arquitectura del proyecto

La arquitectura del proyecto tal y como se dijo en la introducción de este capítulo es la Modelo-Vista-Controlador que en Django se le conoce como Modelo-Plantilla-Vista (Model-Template-View), que en esencia es la misma arquitectura pero con alguna que otra peculiaridad.

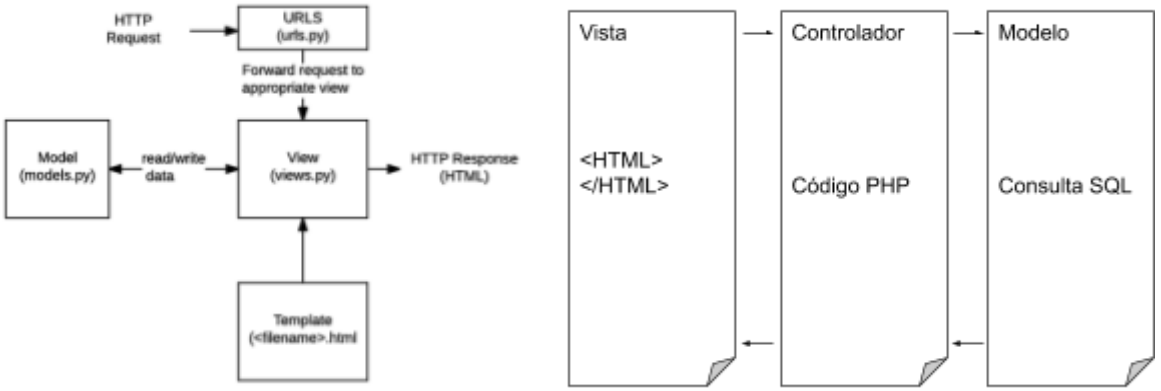


Figura 31. Esquemas del funcionamiento de MTV y MVC

Lo que se conoce como vista dentro del MVC que usualmente en web es un fichero html en Django es conocido como **plantilla** (template), una plantilla tiene el mismo propósito que la vista en MVC y se puede usar para definir la estructura de cualquier tipo de fichero junto a ciertas etiquetas propias de Django para crear una página dinámicamente a través de una *vista* con datos obtenidos de un *modelo*.
Aunque las plantillas no corresponden a un tipo de fichero único, normalmente son ficheros html con etiquetas de Django.

Respecto a la capa controlador en Django es conocida como **vista** (view), una función de Python que gestiona las peticiones HTTP y devuelve una respuesta. Estas vistas acceden a los datos para cumplir con las peticiones a través de los *modelos* y envían dichos datos de respuesta a las *plantillas* para mostrarselos al usuario según un formato concreto. La gestión de estas vistas que renderizan las *plantillas* quedan a cargo del comentado en el capítulo anterior fichero de las urls, que a partir de una petición HTTP la reenvía a la vista correspondiente con la que se va a gestionar.

Por último, tendríamos el **modelo** (model) que en Django se llamaría igual, estos son objetos de Python que definen la estructura de los datos de las entidades y proporcionan mecanismos para añadir, actualizar, borrar y consultar registros de los mismos en una base de datos.

Durante este capítulo se ha llamado a la aplicación principal como proyecto siguiendo la terminología del framework, puesto que un proyecto en Django está compuesto de varias aplicaciones. Una aplicación en Django son partes del software final que tienen

cierta independencia entre sí y desembocan una tarea concreta siguiendo todas el MTV con sus propios modelos, vistas, plantillas y urls.

6.3. Implementación de un caso de uso

Para visualizar lo anterior comentado en la anterior sección a nivel de práctica se mostrará cómo se integró uno de los casos de uso más completo de la aplicación en esta arquitectura, el caso de uso *Registrarse* del rol de invitado. Este proyecto está conformado por tres aplicaciones: **gestionUsuarios**, **gestionArticulos** y **gestionMensajeria**.

Como se dijo anteriormente, cada aplicación desempeña una finalidad distinta según sus propios modelos, sus propias vistas y sus propias plantillas. Dentro de la aplicación de **gestionUsuarios** irían únicamente aquellos recursos relacionados con los casos de uso de la gestión de las cuentas de usuario y consulta de su información. La aplicación **gestionArticulos** por otra parte, comprendería aquellos recursos referentes a los casos de uso de la gestión o consulta de los artículos de los usuarios. Finalmente, tendríamos la aplicación **gestionMensajeria** que destinaría sus recursos a aquellos casos de uso que conforman la comunicación por medio de chats entre usuarios acerca de los artículos. Para crear cada una de estas aplicaciones desde Django dentro del proyecto hubo que ejecutar la siguiente instrucción por medio de la terminal.

```
python manage.py startapp appDeProyecto
```

Cuando una app se crea dentro de la carpeta raíz del proyecto se crea una carpeta con el nombre de la aplicación en la que en su interior por defecto incluye los ficheros relacionados con las vistas (views.py), los modelos (models.py) y el fichero de urls (urls.py) encargado de gestionar las vistas de dicha aplicación. Hecho esto para que estas aplicaciones queden registradas dentro del proyecto con sus referencias a sus recursos deben quedar patentes en el apartado de apps instaladas en el fichero de configuración global del proyecto **settings.py** y debe también estar dentro del fichero general **urls.py** del proyecto la referencia al fichero de urls propio de cada aplicación.

```
# Application definition
INSTALLED_APPS = [
       'django.contrib.admin',
       'django.contrib.auth',
       'django.contrib.contenttypes',
       'django.contrib.sessions',
       'django.contrib.messages',
       'django.contrib.staticfiles',
       'widget_tweaks',
        'mathfilters',
       'gestionUsuarios',
       'gestionArticulos',
       'gestionMensajeria',
```

```
]
```

Por último, para dejar todo bien organizado para trabajar con las aplicaciones, dentro de las carpetas static o template creamos una carpeta por cada aplicación para tener organizados los recursos según corresponde. Con esto explicado pasaríamos a la implementación del caso de uso *Registrarse* que forma parte en el proyecto dentro de la aplicación **gestionUsuarios**.

6.3.1. Capa modelo

Para que el caso de uso *Registrarse* tenga lugar son necesarios todos los modelos de datos implementados en la aplicación **gestionUsuarios**. Estos modelos se encuentran en **models.py** dentro de la carpeta de la aplicación donde cada entidad que se crea es un objeto en Python como se comentó durante este capítulo.

```
class UsuarioUbicacion(models.Model):
	usuario=models.OneToOneField(User, on_delete=models.CASCADE)
	latUb=models.FloatField(verbose_name='Latitud de la ubicación')
	lngUb=models.FloatField(verbose_name='Longitud de la ubicación')
	dirUb=models.CharField(max_length=150, null=True)

	def __str__(self):
	return "Usuario: %s con ubicación con latitud %s y longitud %s y
dirección %s" % (self.usuario.username, self.latUb, self.lngUb, self.dirUb)

class UsuarioInfo(models.Model):
	usuario=models.OneToOneField(User, on_delete=models.CASCADE)
	avatar=models.ImageField(upload_to='avatars')
	bio=models.CharField(max_length=150)
	valoracion=models.IntegerField(default=0)
	n_valoraciones=models.IntegerField(default=0)
	fechaNacimiento=models.DateField(default=timezone.now,
verbose_name='Fecha de nacimiento')
	ubicacion=models.EmbeddedField(model_container=UsuarioUbicacion)
	articulos=models.ArrayReferenceField(to=Media,
related_name='mis_articulos', null=True)
	articulos_rec=models.ArrayReferenceField(to=Media,
related_name='rec_articulos', null=True)

	def __str__(self):
	return "Usuario: %s con email %s y fecha de creación %s" %
(self.usuario.username, self.usuario.email, self.usuario.date_joined)
```

Estas dos entidades están compuestas por aquellos datos que el invitado introducirá para registrarse, los datos generales del usuario (UsuarioInfo) y por otra parte su ubicación (UsuarioUbicacion). La entidad que conforma los datos identificativos del usuario figuraría aquí dentro de los *imports* puesto que los usuarios en Django tienen

su propio modelo de datos ya creado en el proyecto por el sistema de autentificación predefinido.

Para que esta capa funcione adecuadamente es necesario que la conexión de la base de datos y su motor estén especificados dentro del fichero de configuración global **settings.py**.

```
# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/#databases
# Configuración de la base de datos que utilizaremos, en nuestro caso una
con MongoDB con el motor djongo para Django
DATABASES = {
    'default': {
        'ENGINE': 'djongo',
        'NAME': 'mediakaanadb',
        'CLIENT': {
            'host':
'mongodb+srv://<user>:<pass>@mediakaanadb-mevxh.gcp.mongodb.net/test?retryW
rites=true&w=majority',
            'port': 27017,
            'username': '*******',
            'password': '*******',
        }
    }
}
```

Las bases de datos no relacionales de **MongoDB**[6] están alojadas en la nube gracias a los servicios que ellos comparten y la gestión para la nuestra en Django corre a cargo de **Djongo**, un conector que traduce instrucciones SQL a instrucciones de MongoDB para gestionar una base de datos no relacional.

Por último, dado que se tienen que crear las entidades para poder posteriormente insertar los datos en la base de datos, ejecutamos una instrucción para comprobar que los modelos de la aplicación se han escrito correctamente y otra para hacer migrar los modelos tal y como los hemos definido a la base de datos.

```
python manage.py makemigrations gestionUsuarios
python manage.py migrate
```

Cada aplicación de un proyecto tiene sus propios modelos por lo que es necesario migrar cada una de ellas para crear todas las entidades del proyecto en la base de datos. Dentro de esta capa no haría falta más, puesto que Django a través de la definición de los modelos crea internamente los mecanismos para gestionarlos.

[6] Servicio en la nube para bases de datos no relacionales con MongoDB: https://www.mongodb.com/

6.3.2. Capa plantilla

Esta sección al igual que la capa del proyecto correspondiente precisa de dos partes, la de **formularios** que establece cada control con sus validaciones asociadas a los atributos de un modelo y la de **plantilla** dedicada a renderizar aquellos controles sometidos a validaciones con los que el usuario interaccionará.

6.3.2.1. Formularios

Para el caso de uso de *Registrarse* antes de comenzar a crear su plantilla correspondiente se crearon sus controles del formulario por medio del fichero **forms.py** dentro de la aplicación. Los formularios en Django tienen la finalidad de que la definición de los controles esté separada de la **capa plantilla** (en la cual serán renderizados para el usuario) y por otra parte someter a los controles a las validaciones de sus datos según sus validadores definidos, haciendo que la **capa vista** únicamente se encargue de crear o no los objetos con los datos proporcionados por el usuario en la base de datos si las validaciones de los controles han sido correctas.

```
class UsuarioForm(forms.ModelForm):
	"""
	Formulario del usuario.
	"""

	# Texto de ayuda para los campos del formulario
	def __init__(self, *args, **kwargs):
		super(UsuarioForm, self).__init__(*args, **kwargs)
		self.fields['password'].help_text = 'La contraseña debe tener
al menos una letra o número y una longitud como mínimo con 8 caracteres.'

	# Campo de la contraseña con validador de su expresión regular y
mensaje de error
	password = forms.CharField(
	widget=forms.PasswordInput(),
	validators=[
		RegexValidator(
		regex='^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$',
		message='La contraseña debe tener mínimo 8 caracteres y al
menos una letra o número.')])

	passwordRep = forms.CharField(widget=forms.PasswordInput()) # Campo
para repetir la contraseña

	# Validador de la repetición de contraseña con mensaje de error
	def clean_passwordRep(self):
	   password = self.cleaned_data.get('password', False)
	   passwordRep = self.cleaned_data.get('passwordRep', False)

	   if password and password != passwordRep:
```

```
            raise forms.ValidationError("Las contraseñas deben
coincidir.")
         return passwordRep

      # Validador de email repetido
      def clean_email(self):
         email = self.cleaned_data.get('email', False)

         if User.objects.filter(email=email).exists():
            raise forms.ValidationError("El email que ha introducido ya
pertenece a un usuario registrado.")
         return email

      class Meta():
         model = User
         fields = ('username','email','password')

class UsuarioInfoForm(forms.ModelForm):
      """
      Formulario de la información adicional del usuario.
      """

      # Texto de ayuda para los campos del formulario
      def __init__(self, *args, **kwargs):
         super(UsuarioInfoForm, self).__init__(*args, **kwargs)
         self.fields['avatar'].help_text = 'El avatar debe ser una imagen
con dimensiones 500x500.'
         self.fields['fechaNacimiento'].help_text = 'El usuario debe tener
18 años para poder utilizar la aplicación.'

      # Campo de biografía con validadores de longitud entre 100 y 150
caracteres con mensajes de error
      bio = forms.CharField(widget=forms.Textarea(),
      validators=[
            MinLengthValidator(
            100,
            message='La biografía del usuario debe tener al menos 100
caracteres.'),
            MaxLengthValidator(
            150,
            message='La biografía del usuario debe tener como máximo 150
caracteres.')])

      # Campo del avatar del usuario con validador de extensión del
archivo y mensaje de error
      avatar = forms.ImageField(widget=forms.FileInput(),
      validators=[
            FileExtensionValidator(
            allowed_extensions=['jpg', 'jpeg', 'png', 'gif'],
```

```
            message='El avatar debe tener una extensión válida de imagen
(JPG, JPEG, PNG, GIF).')
            ])

      # Campo de fecha de nacimiento
      fechaNacimiento = forms.DateField(widget=forms.DateInput())

      # Validador adicional del campo avatar, valida las dimensiones de la
imagen que sube el usuario
      def clean_avatar(self):
          avatar = self.cleaned_data.get('avatar', False)
          width, height = get_image_dimensions(avatar)
             if avatar:
                if height > 500 or width > 500:
                    raise ValidationError("Las dimensiones del avatar deben
ser de 500x500.")
                return avatar
             else:
                raise ValidationError("No se ha encontrado cargada ninguna
imagen.")

      # Validador de fecha de nacimiento, donde el usuario debe tener 18
años
      def clean_fechaNacimiento(self):
         fechaNacimiento = self.cleaned_data.get('fechaNacimiento', False)
         edad = datetime.now().date() - fechaNacimiento
         if edad.days < 18*365 :
             raise forms.ValidationError("La edad del usuario debe ser
igual o mayor a 18 años para usar la aplicación.")
         return fechaNacimiento

      class Meta():
         model = UsuarioInfo
         fields = ('avatar', 'bio', 'fechaNacimiento')

class UsuarioUbicacionForm(forms.ModelForm):
      """
      Formulario de la ubicación del usuario.
      """

      latUb=forms.FloatField(widget=forms.HiddenInput())
      lngUb=forms.FloatField(widget=forms.HiddenInput())

      # Validador de latitud y longitud no vacia y válida
      def clean(self):
         latUb = self.cleaned_data.get('latUb', False)
         lngUb = self.cleaned_data.get('lngUb', False)

      if latUb and lngUb:
```

```
            # La latitud debe estar entre -90 y 90 y la longitud entre
-180 y 180
            if (latUb >= -90.0 and latUb <= 90.0) and (lngUb >= -180.0 and
lngUb <= 180.0):
                return self.cleaned_data
            else:
                raise forms.ValidationError("Ubicación no válida.")

      class Meta():
         model = UsuarioUbicacion
         fields = ('latUb', 'lngUb')
```

Cada formulario en Django va ligado a un modelo de datos al que se le aplican diferentes reglas a los controles que lo forman, en nuestro caso tendríamos tres formularios pues la información del usuario la formaría un modelo para sus datos identificativos, otro para sus datos generales y otro para su ubicación.

En cada uno de estos formularios de los modelos se especifica que tipo de *input* tendrá cada uno de sus controles con el que el invitado introducirá sus datos, los validadores de cada control que irá ligado al modelo con sus mensajes de error correspondientes y los textos de ayuda de cada control del formulario para que el invitado haga un uso correcto al introducir los datos. Estas características se pueden ver en los formularios que nosotros utilizamos para el caso de uso *Registrarse*:

- **UsuarioForm:** Va ligado a los datos identificativos del usuario y por tanto al modelo de autentificación de usuario que viene por defecto creado en Django (especificado en la clase `Meta`). Lo forma cuatro controles, el del **nombre de usuario** (input text), su **email** (input email), su **contraseña** (input password) y la **repetición de la misma** (input password). Posee textos de ayuda para escribir el nombre de usuario que ya vienen con dicho modelo y uno adicional sobre las reglas aplicadas a nuestro campo de contraseña. Sus validadores son tres, uno expresado por una expresión regular para la contraseña del usuario, otro manual para comprobar que ambas contraseñas son idénticas y otro para comprobar si el email no está entre los registros de la base de datos.

- **UsuarioInfoForm:** Va ligado al modelo de los datos generales del usuario. Lo forma 3 controles, el de la **biografía** del usuario (textarea), su **avatar** (input file) y su **fecha de nacimiento** (input date). Posee un texto de ayuda para mencionar que el avatar debe cumplir unas dimensiones de 500x500 y otro para la fecha de nacimiento especificando que el usuario tiene que tener 18 años. En total posee cinco validadores, dos propios del control de la biografía para especificar que debe tener entre 100 y 150 caracteres, otro propio del control avatar para la extensión del fichero de imagen junto a uno manual para comprobar que las dimensiones de la imagen son correctas y otro manual para la fecha de nacimiento para hacer cumplir una restricción de edad de 18 años.

- **UsuarioUbicacionForm:** Ligado con el modelo de ubicación del usuario. Tiene dos controles de tipo oculto para la **latitud** y la **longitud** de la ubicación del usuario y un validador manual para comprobar que estos controles no se salgan de su rango asociado.

6.3.2.2. Plantilla

La plantilla utilizada para el caso de uso *Registrarse* es **signup.html** y se puede encontrar en el directorio **templates/gestionUsuarios** situado en la carpeta raíz del proyecto. Debido a que su número de líneas es demasiado amplio se mostrarán ciertas partes del fichero para concretar cómo está construída la plantilla. La plantilla está fundamentada en etiquetado de html y etiquetado de Django para hacer la página web dinámica junto a otros tipos de componentes que iremos viendo.

El estilo general de la página está definido por medio de atributos de Bootstrap y una hoja de estilo CSS propia para la plantilla de registro. Esta plantilla está estructurada en forma de carta con scroll y con los componentes del formulario dispuestos en grupos de filas.

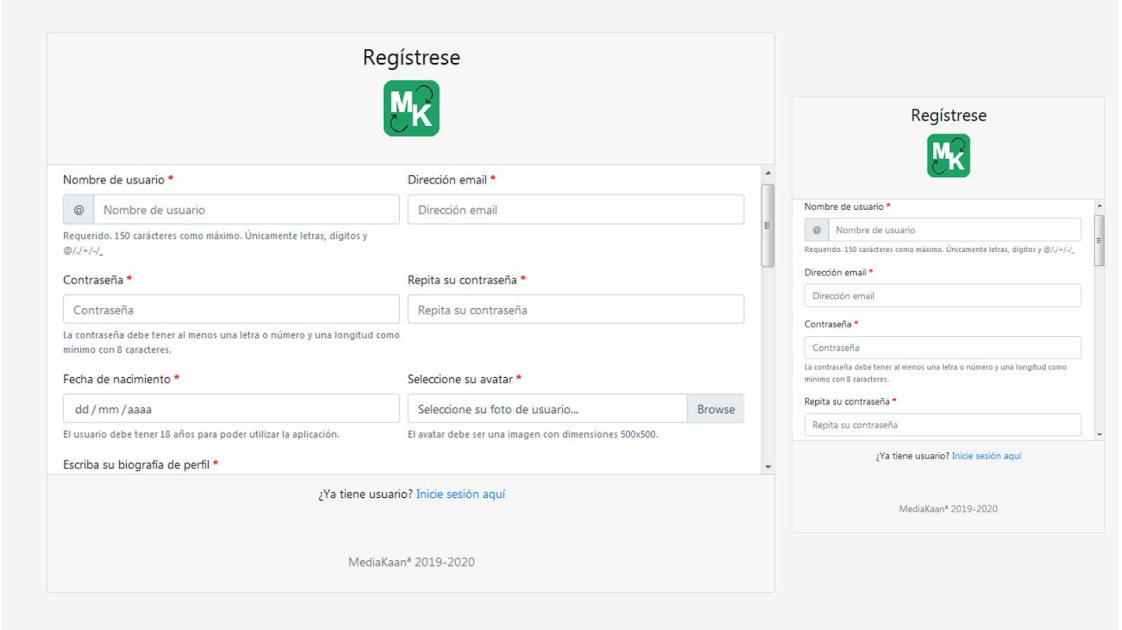


Figura 32. Muestra de la plantilla de registro desde un navegador en PC y desde un navegador móvil

Estos controles que se pueden ver en esta figura son los pertenecientes a los formularios del usuario que vimos anteriormente, para su colocación y estilo en la plantilla hubo que recurrir a la aplicación de Django `widget_tweaks` puesto que sin ella el framework únicamente deja colocar los formularios con sus componentes en formato de tabla o de párrafos html y no deja renderizar cada componente por separado ni especificar sus etiquetas.

```
<div class="col-md-6 mb-3">
   <label for="id_password">Contraseña <strong style="color:
red;">*</strong></label>
```

```
   {% if usuario_form.password.errors %}
   {% render_field usuario_form.password class="form-control is-invalid"
placeholder="Contraseña" required="" %}
   <small class="form-text text-muted" id="helpPassword">
   {{ usuario_form.password.help_text }}
   </small>
   <small class="form-text text-muted" id="errorPassword">
      {% for error in usuario_form.password.errors %}
      <p class="text-danger">{{ error }}</p>
      {% endfor %}
   </small>
   {% else %}
   {% render_field usuario_form.password class="form-control"
placeholder="Contraseña" required="" %}
   <small class="form-text text-muted" id="helpPassword">
   {{ usuario_form.password.help_text }}
   </small>
   {% endif %}
</div>
```

Para la colocación de los controles se devuelven a través de la vista correspondiente los tres formularios del usuario y es a partir de la plantilla donde se renderizan según la estructura que se ha decidido adoptar utilizando sus métodos. Cada control tiene las mismas comprobaciones y está colocado de igual forma que como se puede apreciar con el control contraseña expuesto, a excepción de alguno que luego estudiaremos en detalle.

Primeramente, se accede por medio del formulario al control y comprobamos si ha devuelto errores después de enviar, si ha sido el caso renderizamos el control con un borde rojo para informar al usuario de que los datos no son válidos y con un bucle sacamos los mensajes de error del control para informar al usuario, que serán aquellos que especificamos para cada uno en lo que se vió anteriormente en **forms.py**. Adicionalmente, tanto con el control normal como con el formato invalido renderizamos también su texto de ayuda que especificamos en su lugar correspondiente en el fichero de los formularios.

Por otra parte, los controles colocados de manera excepcional a la utilizada son aquellos referentes a la **ubicación** del usuario y el área de texto de la **biografía**.
Para el área de texto de la **biografía** se definieron tres filas para el mismo, se hizo no redimensionable y se colocó un contador de letras programado en JavaScript con JQuery para ir mostrando al invitado la cantidad que lleva escritas.

El control de la **ubicación** fue algo más complejo, puesto que para que el usuario pudiera introducirla hubo que poner un control para colocar la ubicación dentro de un mapa.

```
<!-- templates/gestionUsuarios/signup.html -->
<div class="form-row">
   {% render_field ubus_form.latUb value="" %}
   {% render_field ubus_form.lngUb value="" %}
   {% include "gestionUsuarios/registerubication.html" %}
   {% if ubus_form.errors %}
   <small class="form-text text-muted" id="errorUb">
      {% for error in ubus_form.latUb.errors %}
      <p class="text-danger">{{ error }}</p>
      {% endfor %}
   </small>
   {% endif %}
</div>
```

```
<!-- templates/gestionUsuarios/registerubication.html -->
<div class="col-md-12 mb-3">
    <label for="ubicacion">Ubicación del usuario <strong style="color:
red;">*</strong></label>
    <small class="form-text text-muted">
        <p>Clique en el botón de "¡Encuéntrame!" y precise clicando en el
mapa su ubicación.</p>
    </small>
    <button type="button" class="btn btn-outline-secondary"
onclick="locateUser()">¡Encuéntrame!</button>
</div>
<div class="col-md-12 mb-3">
    <div id="mapid" style="width: 100%; height: 400px;"></div>
</div>
```

Para hacer esto posible se embebió el fichero html **registerubication.html** dentro de la plantilla de registro para dotar al formulario con un control para definir su ubicación en el mapa a través de un botón que la determine y el puntero del ratón para editarla y hacerla más correcta. Este control se programó con la librería de JavaScript de mapas open source Leaflet[7] y a través de código JQuery se hizo que cada vez que se definiera una nueva ubicación, la latitud y longitud de la ubicación del usuario en el mapa quedará registrada en los controles de tipo oculto del formulario de ubicación del usuario que definimos en el fichero de formularios. Aunque estos controles se coloquen de manera excepcional a lo comentado su colocación de mensajes de error es la misma que todos los demás y siempre se sitúan debajo de los componentes.

[7] Leaflet, Librería de JavaScript de código abierto para mapas interactivos: https://leafletjs.com/

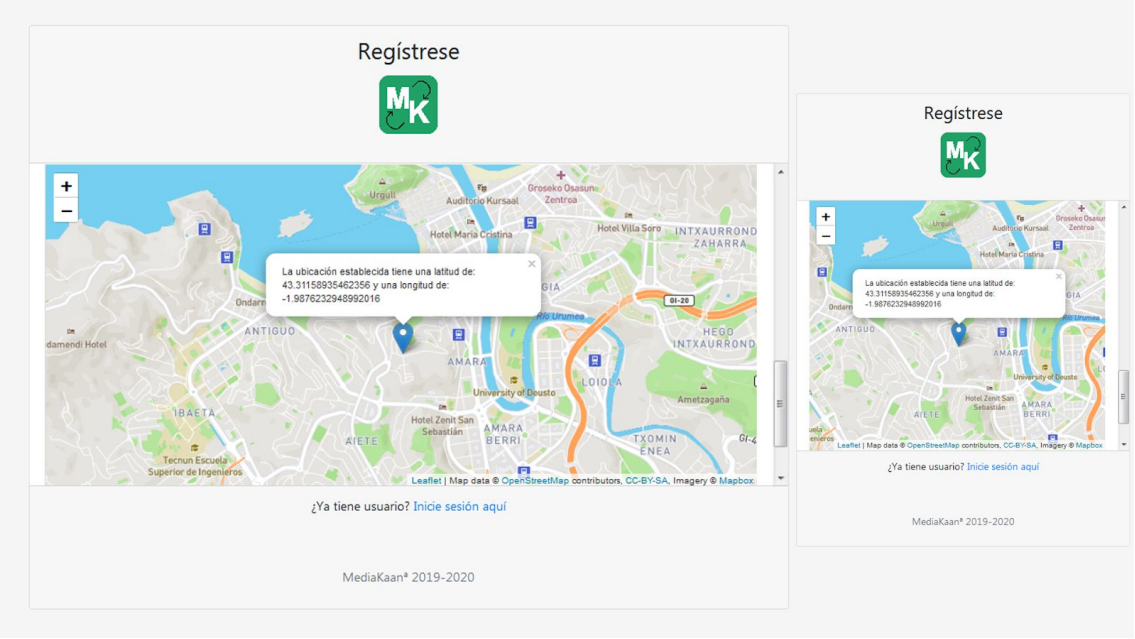


Figura 33. Muestra del control de ubicación de registro en PC y desde un navegador móvil

Con esto explicado faltaría comentar el comportamiento de la plantilla en el caso de que el invitado logre registrarse exitosamente como usuario tras rellenar adecuadamente el formulario.

```
<div class="card-body scroll">
{% if registered %}
<h1>¡Te has registrado con éxito!</h1>
<div class="alert alert-success" role="alert">
   <h4 class="alert-heading" style="padding-bottom: 20;">¡Te has registrado
con éxito!</h4>
   <p>Confirme su cuenta desde su correo e inicie sesión desde
<a href="{% url 'gestionUsuarios:user_login' %}">aquí.</a></p>
</div>
{% else %}
<form>
 <!-- Formulario de registro -->
</form>
{% endif %}
</div>
```

Para hacer este tipo de comprobación dentro de la plantilla, comprobaremos el valor de un booleano que la vista asociada evalúa si el formulario html se ha rellenado correctamente. Cuando el formulario se envía, la vista evalúa y devuelve un booleano a la plantilla de registro para informar de que el registro ha sido exitoso o que el registro no se ha completado correctamente, marcando cada control con su mensaje de error correspondiente.

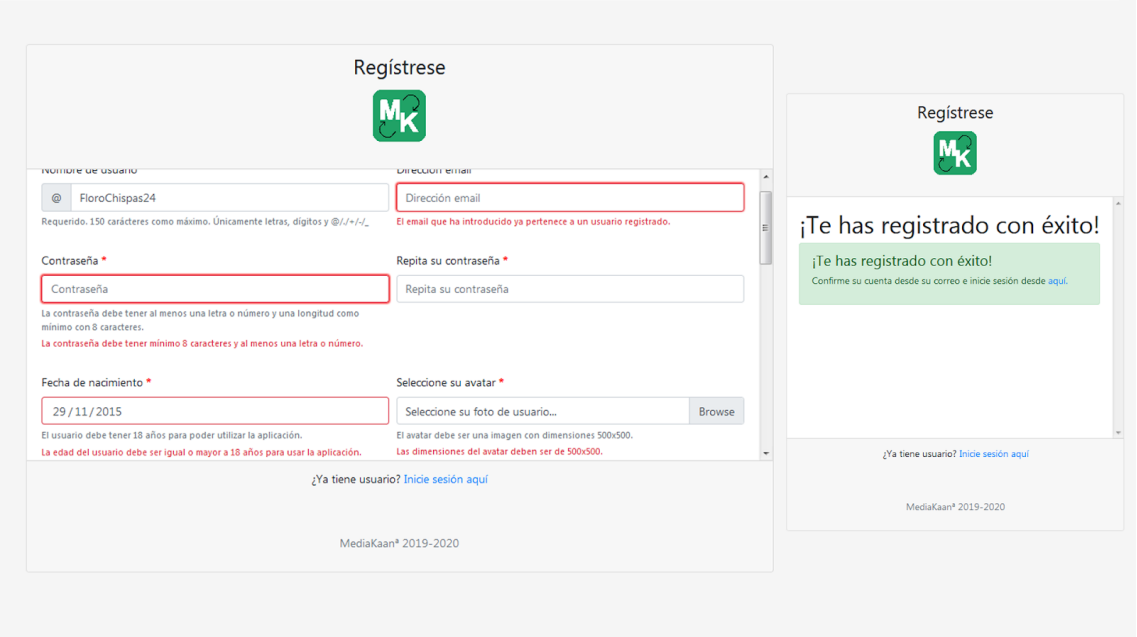


Figura 34. Interfaz registro cuando se envía cometiendo errores y cuando se envía con sus datos satisfactorios

Por último para que esta plantilla funcione adecuadamente con todo lo que tiene, sus referencias a los recursos estáticos en **static/gestionUsuarios** se hacen necesarias para importar las librerías, hojas de estilo o imágenes utilizadas para la plantilla. Algunas de ellas como las de las hojas de estilo se pueden ver al final de la cabecera del fichero html y las librerías de javascript utilizadas se pueden ver en el final de su cuerpo.

```
<!DOCTYPE html>
<html lang="es">
   {% load static %}
   {% load widget_tweaks %}
   <head>
      <!-- ... -->
      <link rel="icon" href="{% static
"gestionUsuarios/img/logo/mediak-logo.ico" %}">
      <title>MediaKaanª | Registrarse</title>
      {% block css %}
      <link href="{% static "gestionUsuarios/css/bootstrap.css" %}"
rel="stylesheet">
      <link href="{% static "gestionUsuarios/css/signin.css" %}"
rel="stylesheet">
      <link rel="stylesheet"
href="https://unpkg.com/leaflet@1.6.0/dist/leaflet.css"

integrity="sha512-xwE/Az9zrjBIphAcBb3F6JVqxf46+CDLwfLMHloNu6KEQCAWi6HcDUbeO
fBIptF7tcCzusKFjFw2yuvEpDL9wQ=="
         crossorigin="" />
      {% endblock %}
```

```
   </head>
   <body>
      <!-- ... -->
      {% block javascript %}
      <script type="text/javascript" src="{% static
"gestionUsuarios/js/jquery-3.4.1.min.js" %}"></script>
      <script type="text/javascript"
src="https://unpkg.com/leaflet@1.6.0/dist/leaflet.js"

integrity="sha512-gZwIG9x3wUXg2hdXF6+rVkLF/0Vi9U8D2Ntg4Ga5I5BZpVkVxlJWbSQtX
PSiUTtC0TjtGOmxa1AJPuV0CPthew=="
         crossorigin=""></script>
      {% endblock %}
      {% block extra_js %}
      <script type="text/javascript" src="{% static
"gestionUsuarios/js/signupcounter.js" %}"></script>
      <script type="text/javascript" src="{% static
"gestionUsuarios/js/registerubication.js" %}"></script>
      {% endblock %}
   </body>
</html>
```

6.3.3. Capa vista

La vista correspondiente al caso de uso *Registrarse* está dentro del fichero **views.py** de la aplicación de **gestionUsuarios**. Cada vista en este fichero es un método encargado de renderizar una plantilla parametrizada, en nuestro caso esta vista es el método llamado **register**. Para que esta vista sea capaz de funcionar lo primero que tenemos que hacer es registrarla junto a su url en el fichero **urls.py** de la propia aplicación, para que este se encargue de redirigir la petición http a la vista correspondiente.

```
from django.conf.urls import url
from . import views

app_name = 'gestionUsuarios'

urlpatterns = [
      url(r'^registro/$', views.register, name='register'),
      url(r'^iniciosesion/$', views.user_login, name='user_login'),
url(r'^activar/(?P<uidb64>[0-9A-Za-z_\-]+)/(?P<token>[0-9A-Za-z]{1,13}-[0-9
A-Za-z]{1,20})/$', views.activate, name='activate'),
      url(r'^olvidopass/$', views.forget_pass, name='forget_pass'),
url(r'^cambiopass/(?P<uidb64>[0-9A-Za-z_\-]+)/(?P<token>[0-9A-Za-z]{1,13}-[
0-9A-Za-z]{1,20})/$', views.change_pass, name='change_pass'),
      url(r'^miperfil/$', views.my_profile, name='my_profile'),
      url(r'^eliminarperfil/$', views.delete_user, name='delete_user'),
      url(r'^resultados/$', views.users_results, name='usrresults'),
      url(r'^usuario/$', views.user, name='usr'),
```

```
]
```

La vista correspondiente al caso de uso de *Registrarse* está implementada de la siguiente forma.

```
def register(request):
      registered = False
      if request.method == 'POST':
      usuario_form = UsuarioForm(data=request.POST) # Tomamos el
formulario de los datos identificativos del usuario
      infous_form = UsuarioInfoForm(data=request.POST,
files=request.FILES)
# Tomamos el formulario de los datos adicionales del usuario
      ubus_form = UsuarioUbicacionForm(data=request.POST) # Tomamos el
formulario de la ubicación del usuario

      # Comprobamos que todos los formularios son correctos, si no son
validos la capa cliente
      # mostrará los errores correspondientes de cada campo del
formulario.
      if usuario_form.is_valid() and infous_form.is_valid() and
ubus_form.is_valid():
            # Creación del usuario
            usuario = usuario_form.save()
            usuario.set_password(usuario.password)
            usuario.is_active = False # Dejamos desactivada la cuenta para
que el usuario la active por medio de un email
            usuario.save()

            # Creación de la ubicación del usuario
            ubus = ubus_form.save(commit=False)
            ubus.usuario = usuario
            geolocator = Nominatim(user_agent="MediaKaan")
            location = geolocator.reverse(str(ubus.latUb) + ", " +
str(ubus.lngUb))
            direccion = ', '.join(location.address.split(',', 4)[1:4])
            ubus.dirUb = direccion
            ubus.save()

            # Creación del perfil de información del usuario
            infous = infous_form.save(commit=False)
            infous.usuario = usuario
            infous.avatar = request.FILES['avatar']
            infous.ubicacion = ubus
            infous.save()

            # Envío del email para activar la cuenta
            activation_email(request, usuario, usuario_form)
```

```
            registered = True
      else:
            print(usuario_form.errors, infous_form.errors,
ubus_form.errors)
      else:
      usuario_form = UsuarioForm()
      infous_form = UsuarioInfoForm()
      ubus_form = UsuarioUbicacionForm()

      return render(request,'gestionUsuarios/signup.html',
                        {'usuario_form': usuario_form,
                        'infous_form': infous_form,
                        'ubus_form': ubus_form,
                        'registered': registered})
```

Inicialmente, se crea la variable ***registered*** evaluada a falso pues aún no se han hecho comprobaciones de los formularios, después de ello tomamos los formularios con los datos proporcionados por el invitado y comprobamos que los tres han sido validados satisfactoriamente y si este no ha sido el caso, la vista enviará a la plantilla los formularios tal y como se encuentran con los errores en sus campos junto a sus mensajes.

Cuando estos tres formularios de los datos del usuario (identificativos, generales y de ubicación) sean satisfactorios, a partir de un método de los formularios crearemos los objetos correspondientes a sus modelos de datos e insertaremos dichos registros en la base de datos de **MongoDB** soportada en la nube. Aquí hay un punto a tener en cuenta y es que en el modelo de usuarios propio de Django los usuarios por defecto tienen la variable de activación de la cuenta evaluada a verdadero y puesto que el caso de uso de *Registrarse* necesita completarse con el de *Confirmar usuario* antes de registrar el objeto con los datos identificativos del usuario, pondremos su variable a falso. Otra cosa a tener en cuenta es que el campo dirección del objeto ubicación es asignado con texto sobre la ubicación real a partir de la latitud y longitud que nos proporcionan los controles de **Leaflet** utilizados por el usuario en la plantilla, para realizar dicha asignación de la dirección desde la vista utilizamos un geocodificador que nos ofrece la dirección de la api **GeoPy**[8], una librería de Python de código abierto para servicios de geolocalización. Dicho objeto ubicación en ese punto además la embebemos dentro del campo ubicación del objeto de información adicional del usuario para así realizar menos consultas a la base de datos en los distintos casos de uso que utilicen esa información.

Una vez estos tres objetos hayan sido registrado en la base de datos, el siguiente paso será enviar el correo electrónico de activación de la cuenta de usuario al invitado.
Esta parte del código para enviar el email se ejecuta a partir de nuestra vista en **gestionUsuarios\emails.py**, un fichero con los métodos de los distintos correos que se envían en los diferentes casos de uso para informar al usuario, una característica

[8] GeoPy, librería de Python opensource para geolocalización: https://geopy.readthedocs.io/en/stable/

bastante utilizada en la aplicación. Para este mensaje inicialmente tomamos la dirección de dominio del proyecto y escribimos el asunto del mensaje. Puesto que el punto importante aquí es el cuerpo del mensaje y es necesario que sea claro para que el invitado acabe de crear su cuenta de usuario a través del caso de *Confirmar usuario*, hacemos que este sea de tipo html.
Respetando la arquitectura de Django en este punto creamos el cuerpo del mensaje a partir de la plantilla **gestionUsuarios\email\email_active.html** renderizada a una cadena de caracteres con una serie de parámetros de Django.

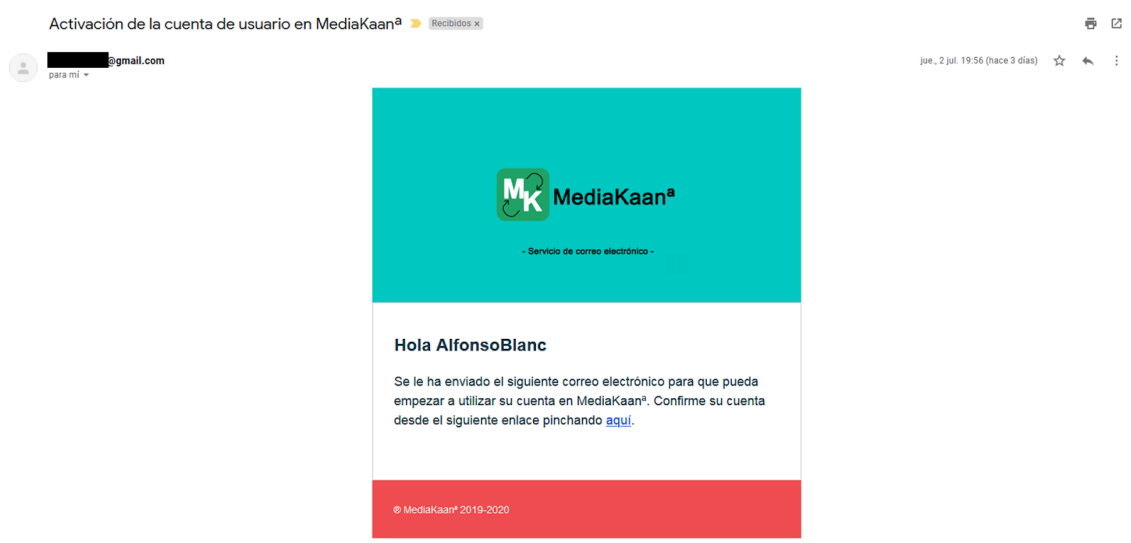


Figura 35. Plantilla renderizada del cuerpo del email de confirmación de una cuenta

Los parámetros utilizados son por un lado el **usuario** (con el que tomar su nombre para hacer personal el mensaje), el **dominio** (con el que construir la url correspondiente a la vista de activación de la cuenta), el **identificador de usuario codificado** (que irá unido a la url de activación para que quede referenciado con la cuenta de usuario que se quiere activar) y un **token** (para completar la url, que será una clave hash temporal para validar la activación de la cuenta). Para la inclusión de este token dentro de la carpeta de la aplicación **gestionUsuarios** se creó el fichero **tokens.py**, el cual se encarga de crear el token de activación de la cuenta por medio de un hasheo del identificador del usuario, con una marca de tiempo específica y con el valor actual de su variable de activación.

```
from django.contrib.auth.tokens import PasswordResetTokenGenerator
from django.utils import six

class TokenGenerator(PasswordResetTokenGenerator):

    def _make_hash_value(self, usuario, timestamp):
        return (
            six.text_type(usuario.pk) + six.text_type(timestamp) +
            six.text_type(usuario.is_active)
        )
```

```
account_activation_token = TokenGenerator()
```

Ultimado el cuerpo del mensaje, referenciamos el receptor del mensaje tomando el email del formulario de los datos identificativos del usuario y creando el objeto EmailMessage con el asunto, el cuerpo del mensaje y el receptor lo enviamos.

Para que este envío pueda darse correctamente configuramos el servicio de correo y registramos el correo electrónico con su contraseña que utilizaremos para enviar el mensaje en el fichero de configuración global del proyecto **settings.py**.

```
# Para informar a través de los email
EMAIL_BACKEND='django.core.mail.backends.smtp.EmailBackend'
EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = '**********@gmail.com'
EMAIL_HOST_PASSWORD = '**********'
EMAIL_PORT = 587
```

Por último dado que todo se ha realizado correctamente, evaluamos la variable registered a verdadero y renderizamos la plantilla de registro haciendo que a partir de esta variable informe al invitado de que su registro como usuario ha sido exitoso.

7. Pruebas

Las pruebas realizadas en una aplicación es un punto de vital importancia sobre todo en lo que es su punto más álgido que es su lanzamiento o su salida a producción. Este es un momento clave pues si no se han realizado las pruebas adecuadas se pueden encontrar vulnerabilidades de seguridad de día cero[9], algo que en la etapa de producción de una aplicación puede ser tortuoso de arreglar y puede ser aprovechado por agentes externos.

Este capítulo reúne la información de las pruebas que se han realizado a lo largo del desarrollo de la aplicación pasando por las realizadas en local, las realizadas en la nube, las de adaptabilidad de la interfaz en dispositivos y finalmente las probadas con usuarios reales.

7.1. Pruebas de las funcionalidades en local

Inicialmente un proyecto de Django tiene en su fichero de configuración global **settings.py** la variable DEBUG evaluada a verdadero, la cual sirve en este estado para comprobar las trazas de errores detectados en el código desde la parte del cliente para depurarlos. Durante todo el desarrollo en local se ha dejado esta variable en tal estado para detectar y solucionar los máximos posibles dentro de la lógica de la aplicación y por otra parte se ha alterado o vaciado la base de datos una gran cantidad de veces para también intentar dar con errores de diseño.

Tras una evaluación final en el estado actual de la aplicación en local, se muestran a continuación algunas pruebas de caja negra con los resultados obtenidos.

Pruebas de invitado en registro e inicio de sesión (figura 3 y figura 5)

- **Prueba 1:** El invitado deja en la plantilla registro los campos biografía y nombre de usuario sin rellenar.
    - **Entrada:** Campo de biografía y nombre de usuario vacíos.
    - **Salida esperada:** Componentes con borde rojo avisando de que los campos sin escribir son obligatorios.
    - **Salida obtenida:** Igual a la salida esperada.

- **Prueba 2:** El invitado rellena la plantilla de registro con un email que ya existe en la aplicación y con una contraseña que no respeta su texto de ayuda.
    - **Entrada:** Campo de email con email existente y contraseña mal escrita.
    - **Salida esperada:** Texto de error debajo del componente email y componente contraseña.
    - **Salida obtenida:** Igual a la salida esperada.

[9] El término inglés zero-day breaches dentro de la informática hace referencia a aquellas vulnerabilidades no detectadas por los desarrolladores con una aplicación en fase de producción, por lo que tienen cero días sin haber sido detectadas y solucionadas desde que se lanzó la aplicación.

- **Prueba 3:** El invitado rellena la plantilla de registro con todos los campos obligatorios escritos correctamente.
    - **Entrada:** Campos obligatorios correctamente escritos.
    - **Salida esperada:** Mensaje de registro exitoso.
    - **Salida obtenida:** Igual a la salida esperada.

- **Prueba 4:** El invitado rellena la plantilla de inicio de sesión con sus datos correctamente escritos sin haber confirmado su cuenta de usuario.
    - **Entrada:** Campos usuario y contraseña bien escritos sin confirmar cuenta.
    - **Salida esperada:** Mensaje de advertencia sobre cuenta de usuario no activada.
    - **Salida obtenida:** Igual a la salida esperada.

- **Prueba 5:** El invitado rellena la plantilla de inicio de sesión con sus datos de usuario escritos de forma incorrecta.
    - **Entrada:** Campos usuario y contraseña rellenados incorrectamente.
    - **Salida esperada:** Mensaje de error de inicio de sesión sobre el usuario y/o la contraseña escritos de forma incorrecta.
    - **Salida obtenida:** Igual a la salida esperada.

- **Prueba 6:** El invitado rellena la plantilla de inicio de sesión con sus datos de usuario escritos correctamente.
    - **Entrada:** Campos usuario y contraseña rellenados correctamente.
    - **Salida esperada:** Redirección a plantilla de inicio con un saludo para el usuario.
    - **Salida obtenida:** Igual a la salida esperada.

Pruebas de usuario con artículos del usuario y gestión de cuenta (figura 19 y figura 14)

- **Prueba 1:** El usuario clica en el apartado de sus artículos.
    - **Entrada:** Click en el enlace de artículos del usuario.
    - **Salida esperada:** Redirección a plantilla de artículos del usuario sin artículos registrados o con ellos.
    - **Salida obtenida:** Igual a la salida esperada.

- **Prueba 2:** El invitado accede por url a la plantilla de artículos del usuario.
    - **Entrada:** URL de la vista de artículos del usuario.
    - **Salida esperada:** Redirección a plantilla de inicio de sesión sin dejar acceder al invitado a una parte restringida para solo usuarios.
    - **Salida obtenida:** Igual a la salida esperada.

- **Prueba 3:** El usuario asigna uno de sus artículos en la plantilla de artículos del usuario a un nombre de usuario inexistente en la aplicación.
    - **Entrada:** Campo de asignado de uno de los artículos del usuario con nombre de usuario inexistente.

    - **Salida esperada:** Mensaje de error de asignación del artículo.
    - **Salida obtenida:** Igual a la salida esperada.

- **Prueba 4:** El usuario asigna uno de sus artículos en la plantilla de artículos del usuario a un nombre de usuario existente en la aplicación.
    - **Entrada:** Campo de asignado de uno de los artículos del usuario con nombre de usuario existente.
    - **Salida esperada:** Artículo listado en plantilla de artículos del usuario con el nombre del usuario al que se le ha asignado.
    - **Salida obtenida:** Igual a la salida esperada.

- **Prueba 5:** El usuario clica borrar en uno de sus artículos en la plantilla de artículos del usuario.
    - **Entrada:** Click en botón borrar artículo.
    - **Salida esperada:** Mensaje de éxito en el borrado del artículo con plantilla de artículos del usuario sin el artículo borrado.
    - **Salida obtenida:** Igual a la salida esperada.

- **Prueba 6:** El usuario actualiza su fecha de nacimiento desde la plantilla de su perfil con una fecha incorrecta.
    - **Entrada:** Campo de fecha de nacimiento con fecha de menos de 18 años.
    - **Salida esperada:** Mensaje de error debajo de la fecha de nacimiento advirtiendo que el usuario debe ser mayor de edad.
    - **Salida obtenida:** Igual a la salida esperada.

- **Prueba 7:** El usuario actualiza su fecha de nacimiento desde la plantilla de su perfil con una fecha correcta.
    - **Entrada:** Campo de fecha de nacimiento con fecha de al menos 18 años.
    - **Salida esperada:** Mensaje exitoso de actualización del perfil de usuario.
    - **Salida obtenida:** Igual a la salida esperada.

7.2. Pruebas de las funcionalidades en la nube

Para realizar pruebas en la nube simulando la fase de producción de la aplicación, esta se ha alojado en un host proporcionado de forma gratuita por **PythonAnywhere**[10] durante un período de tiempo controlado en la url fostergun.pythonanywhere.com. Para hacer funcionar dicha aplicación en la nube se ha utilizado una de las terminales en Linux que nos proporciona este servicio para gestionar nuestra aplicación, para subirla a la nube se hizo una clonación del proyecto en GitHub en el sistema de archivos que nos proporciona el servicio y tras crear un entorno virtual descargamos el software y las librerías necesarias para nuestro proyecto del fichero **requirements.txt**

[10] Sitio especializado en ofrecer hosting de proyectos con Python que además soporta Django: https://www.pythonanywhere.com/

encontrado en la raíz del mismo con el nombre de los paquetes necesarios seguidos de su versión adecuada. Para su instalación se ejecutó el siguiente comando.

```
pip install requirements.txt
```

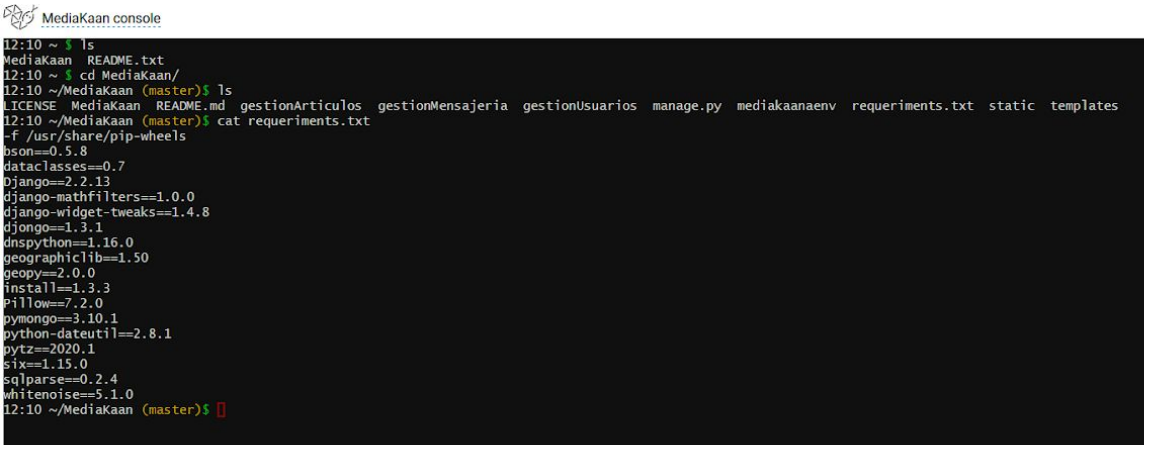


Figura 36. Consola de PythonAnywhere de la aplicación en la nube mostrando el fichero requirements.txt

Hecho eso, configuramos el WSGI[11] de la aplicación junto a un par de carpetas (static y media), evaluamos la variable DEBUG de settings.py a falso para poner la aplicación en modo de producción y colocamos la url de nuestra aplicación en la variable ALLOWED HOSTS para poder probarla tras recargar la aplicación web para que se aplicarán todos los cambios. Si se desean ver en detalle dichas configuraciones junto algunas más como la ruta del entorno virtual utilizado, el código fuente del fichero WSGI o la configuración de la seguridad elegida proporcionada por el servicio de hosting (HTTPS para la aplicación y protección de contraseñas), estas pueden ser consultadas en el **Anexo F**.

Puesto que una aplicación en la nube no es lo mismo que estar alojada en local, aquí se ofrecen algunas pruebas de caja negra que se realizaron sobre algunos casos de uso que precisan del envío de correos electrónicos a usuarios por medio de la aplicación y que además precisan de gestión o consulta de la base de datos de MongoDB en la nube.

[11] En inglés Web Server Gateway Interface, es un tipo de fichero estándar de Python que especifica cómo se comunica un servidor web con la aplicación web, sirve principalmente para que el servidor entienda cómo ejecutar las aplicaciones web que están soportadas con Python (Aplicaciones de DJango, Flask...). https://www.python.org/dev/peps/pep-3333/

Pruebas de envío de correo en confirmar registro, recuperar cuenta y enviar sugerencia (figura 4, figura 6 y figura 12)

- **Prueba 1:** El usuario finaliza su registro exitosamente y recibe un correo de activación.
    - **Entrada:** Finalización de registro exitoso y comprobación de su correo.
    - **Salida esperada:** Notificación nueva en el correo electrónico con un enlace de activación de cuenta en el dominio utilizado.
    - **Salida obtenida:** Igual a la salida esperada.

- **Prueba 2:** El usuario rellena en la plantilla de olvidar contraseña el campo del correo electrónico incorrectamente.
    - **Entrada:** Campo de correo electrónico en la plantilla de olvidar contraseña rellenado incorrectamente con un email inexistente en los registros de la aplicación.
    - **Salida esperada:** Mensaje de error de envío de correo de recuperación en plantilla de olvidar contraseña.
    - **Salida obtenida:** Igual a la salida esperada.

- **Prueba 3:** El usuario rellena en la plantilla de olvidar contraseña el campo del correo electrónico correctamente.
    - **Entrada:** Campo de correo electrónico en plantilla de olvidar contraseña con correo existente en los registros de la aplicación.
    - **Salida esperada:** Mensaje de éxito en envío de correo de recuperación y nueva notificación en el correo electrónico con el enlace de recuperación en el dominio.
    - **Salida obtenida:** Igual a la salida esperada.

- **Prueba 4:** El usuario rellena inadecuadamente los campos de nueva contraseña en la plantilla de cambiar contraseña a través del enlace recibido en su correo electrónico de recuperar cuenta.
    - **Entrada:** Campos de nueva contraseña rellenados incorrectamente.
    - **Salida esperada:** Mensaje de error debajo de los campos mal rellenados.
    - **Salida obtenida:** Igual a la salida esperada.

- **Prueba 5:** El usuario rellena adecuadamente los campos de nueva contraseña en la plantilla de cambiar contraseña a través del enlace recibido en su correo electrónico de recuperar cuenta.
    - **Entrada:** Campos de nueva contraseña rellenados correctamente.
    - **Salida esperada:** Mensaje de éxito al cambiar la contraseña y contraseña actualizada del usuario.
    - **Salida obtenida:** Igual a la salida esperada.

- **Prueba 6:** El usuario rellena en la plantilla de acerca de un mensaje de sugerencia sin los caracteres necesarios.
    - **Entrada:** Campo de sugerencia con caracteres insuficientes.

- **Salida esperada:** Mensaje de error sobre el envío de la sugerencia debajo de su campo.
- **Salida obtenida:** Igual a la salida esperada.

- **Prueba 7:** El usuario rellena en la plantilla de acerca de un mensaje de sugerencia con los caracteres necesarios.
  - **Entrada:** Campo de sugerencia rellenado correctamente.
  - **Salida esperada:** Mensaje de éxito al enviar sugerencia y nueva notificación en el correo electrónico del desarrollador referenciando la autoría del usuario con su sugerencia.
  - **Salida obtenida:** Igual a la salida esperada.

7.3. Pruebas de interfaz en distintos dispositivos

Debido a que nuestra aplicación se presenta como multiplataforma las pruebas de adaptabilidad de la interfaz en dispositivos con distintas resoluciones es algo a considerar para comprobar si todos los controles se adaptan adecuadamente. Estas pruebas se han ido realizando a lo largo del desarrollo con todas las interfaces de la aplicación probando las diferentes resoluciones y su resultado final ha resultado en todas ellas ser positivo.

Como ejemplo, la siguiente figura muestra la adaptabilidad de la interfaz de la página de *Presentación y créditos* vista en dos dispositivos con resoluciones diferentes.

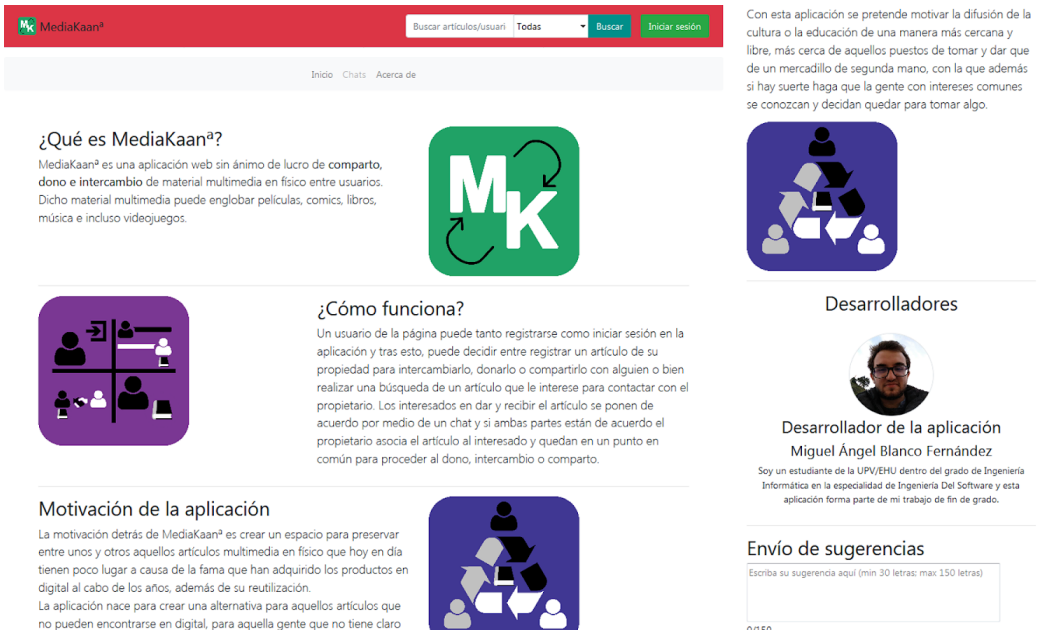


Figura 37. Interfaz de Acerca de desde PC y móvil

En cuanto a las demás pruebas de interfaz, el resto de interfaces se ha mostrado en las figuras del capítulo de *Análisis y diseño* para facilitar la comprensión del flujo de eventos de los casos de uso de dicho capítulo.

7.4. Pruebas con usuarios reales

Por último, hemos realizado pruebas con tres usuarios reales preguntando acerca de sus experiencias con la aplicación. Su experiencia con la aplicación ha sido de una valoración media de **muy satisfecho** con algunos comentarios sobre mejoras en los que han coincidido todos.

- *"El registro podría ser más inmediato dando la opción de hacerlo por Google."*
- *"Al buscar usuarios o artículos, una lista de autocompletado mientras escribes mejoraría las consultas."*
- *"La parte de conversaciones podría mostrar más información del artículo o del propietario."*
- *"Al asignar un artículo a un usuario con su nombre, una lista de sugerencias de usuarios mejoraría la asignación."*
- *"El apartado de conversaciones está bien pero podría ser más intuitivo y claro."*

Una de las conclusiones sacadas de estos comentarios es que habría maneras de mejorar la aplicación introduciendo **un apartado de conversaciones mejorado que sea más claro y más informativo**, **sugerencias de búsqueda en tiempo real** y **registros de usuario alternativos** (cuenta de Google, Facebook, Twitter...).

8. Gestión y seguimiento

En este capítulo veremos cómo se ha establecido la metodología de trabajo, las herramientas utilizadas para planificar el proyecto y el seguimiento y control respecto a lo planificado.

8.1. Metodología

Durante el desarrollo de este proyecto se planteó utilizar la metodología ágil de trabajo Scrum. Sin embargo, dado que esta clase de metodologías están pensadas para coordinar trabajos grupales de forma efectiva (con su sistema de roles correspondiente por integrante), esto hacía que no fuera una metodología adecuada para nuestro proyecto.

Por otra parte, puesto que algunas herramientas de la metodología Scrum se veían que podrían resultar útiles para la realización del proyecto al haberse estudiado en el grado, se utilizaron algunos de sus recursos para la gestión y seguimiento del proyecto como el **sistema de sprints** o el **control diario de dedicaciones**. Lo que nos permitiría balancear cargas de trabajo, organizar el trabajo de las funcionalidades según su importancia y realizar una monitorización y control del tiempo dedicado a cada parte del proyecto.

8.2. Planificación inicial frente a real

Esta sección describe cómo se ha realizado la planificación y cuál ha sido el resultado real bajo la metodología de trabajo comentada en la sección anterior.

8.2.1. Diagrama de Gantt inicial y final

Inicialmente, para la realización del seguimiento y control del proyecto se realizó un primer diagrama de Gantt para la planificación de los sprints necesarios para desarrollar la aplicación y los demás paquetes de trabajo, donde cada sprint estaba establecido con un período de dos semanas con una media de **20 horas por semana** de las **300 totales a planificar**. Este diagrama inicial puede verse en la siguiente tabla.

	08/03	15/03	22/3	29/3	05/04	12/04	19/04	26/04	03/05	10/05	17/05
Aprendizaje e instalación de las tecnologías											
Análisis de requisitos											
Diseño, desarrollo y testing/debug											
Sprint 1											
Registrarse											
Iniciar sesión											
Consultar artículos											
Cerrar sesión											
Confirmar registro											
Casos panel de administrador											
Sprint 2											
Recuperar cuenta											
Consultar perfiles											
Eliminar cuenta											
Editar perfil											
Enviar sugerencia											
Sprint 3											
Registrar artículo											
Asignar artículo											
Eliminar artículo											
Valorar perfiles											
Enviar mensaje											
Depuraciones y pruebas finales											
Memoria del proyecto											
Seguimiento y control											

Tabla 1. Diagrama de Gantt inicial

El diagrama inicial se fijaba entre Marzo y finales de Mayo para toda la realización del proyecto, sin embargo, a causa de la gran cantidad de desvíos que se fueron encontrando durante este período se acabó rechazando, lo que hizo que se tuviera que volver a replantear la estrategia que se había adoptado.

A finales de Mayo se realizó un segundo diagrama de Gantt que fue adoptado como el diagrama final tras unas revisiones realizadas en Junio, reanudando el desarrollo de la aplicación. Este diagrama fijaba el período de los nuevos sprints (de misma duración) a todo el mes de Junio como período de desarrollo y pruebas de la aplicación y por otra parte dos semanas de Julio para la realización de un borrador completo de la memoria, tal y como se puede apreciar en la tabla siguiente.

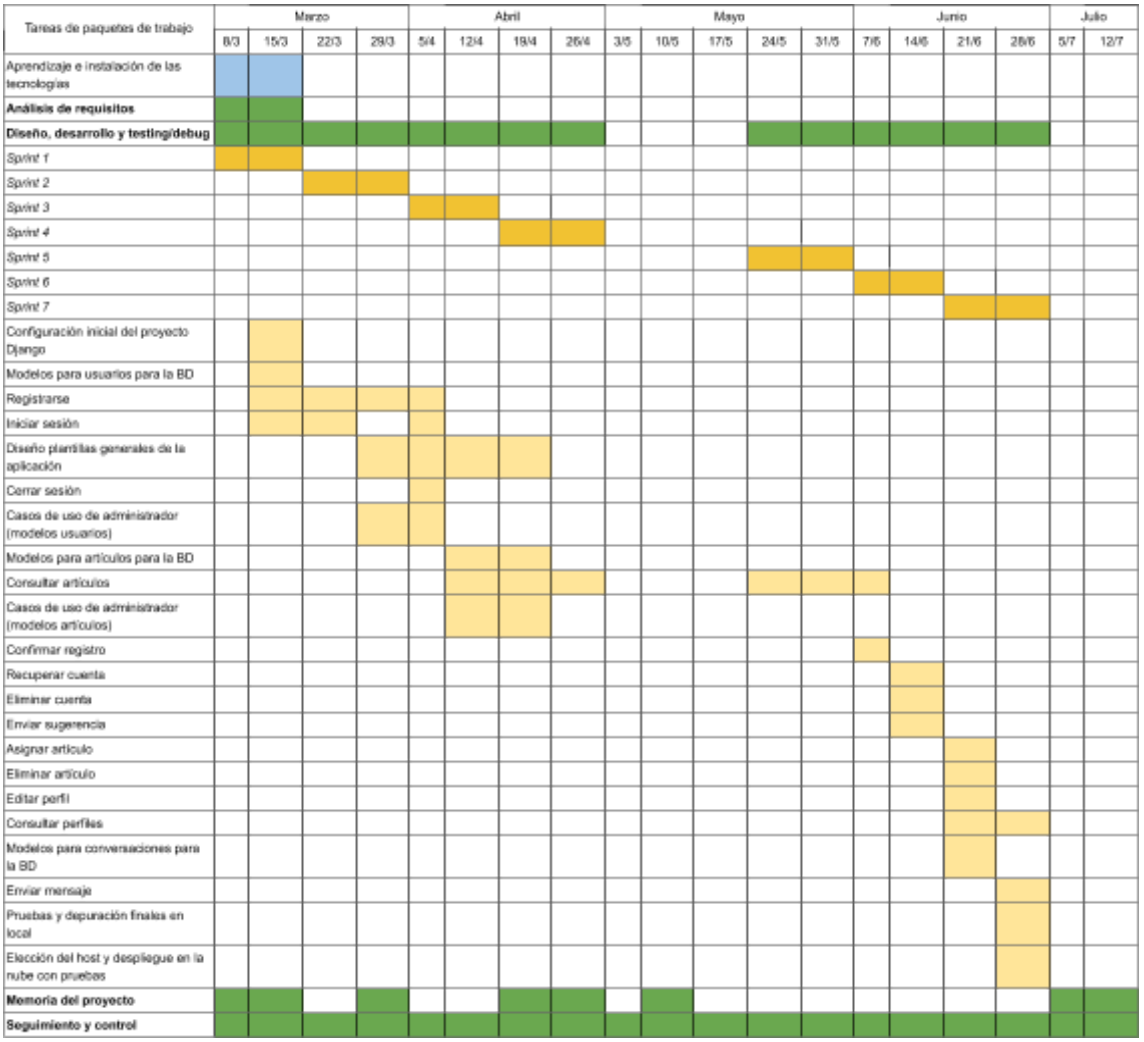


Tabla 2. Diagrama de Gantt final

8.2.2. Dedicaciones esperadas y reales

A medida que se fue trabajando el proyecto, muchas de las horas planificadas distaron respecto a las reales obtenidas por cada paquete de trabajo, quedando las horas planificadas y las horas obtenidas de la siguiente forma.

Paquetes de trabajo	Horas estimadas
AW1. Análisis de requisitos	7
AW2. Diseño de la aplicación	35
AW3. Desarrollo de la aplicación	146
AW4. Testing y depuración	45
Total Aplicación Web	233
MP1. Memoria del proyecto	50
Total Memoria Del Proyecto	50
G1. Planificación	3
G2. Seguimiento y control	2
G3. Aprendizaje organizativo	12
Total Gestión	17
Horas totales	300

Paquetes de trabajo	Horas reales
AW1. Análisis de requisitos	6
AW2. Diseño de la aplicación	28
AW3. Desarrollo de la aplicación	154
AW4. Testing y depuración	38
Total Aplicación Web	226
MP1. Memoria del proyecto	70
Total Memoria Del Proyecto	70
G1. Planificación	4
G2. Seguimiento y control	2
G3. Aprendizaje organizativo	15
Total Gestión	21
Horas totales	317

Tabla 3. Tablas de dedicaciones esperadas y obtenidas

Las horas reales como se pueden apreciar acabaron superando el límite de las planificadas en 17 horas, un resultado que consideramos adecuado dadas las desviaciones y riesgos encontrados durante el desarrollo de todos los paquetes de trabajo del proyecto. Adicionalmente a estos resultados, la siguiente tabla muestra cuál fue el seguimiento de las horas reales por semana.

Semanas	Horas de dedicación reales
08/03	10
15/03	12
22/3	14
29/3	20
5/4	30
12/4	28
19/4	13
26/4	7
3/5	5
10/5	6
17/5	8
24/5	14
31/5	13
7/6	18
14/6	25
21/6	27
28/6	29
5/7	23
12/7	15
Horas totales	317

Tabla 4. Tabla de horas de dedicación por semana

De esta tabla podemos comprobar los incrementos y descensos de horas trabajadas por semana en lo que fue el desarrollo de todo el proyecto, pudiendo ver que una de las zonas que sufrió una disminución considerable fueron las fechas comprendidas entre el **26 de Abril** y el **17 de Mayo**, dando como resultado que a finales de Mayo se tuviera que replantear la planificación de lo que quedaba por realizar del proyecto. Este *valle* de dedicación puede verse de forma más clara desde el gráfico de trabajo de nuestro repositorio de la aplicación proporcionado por GitHub expuesto en la siguiente figura.

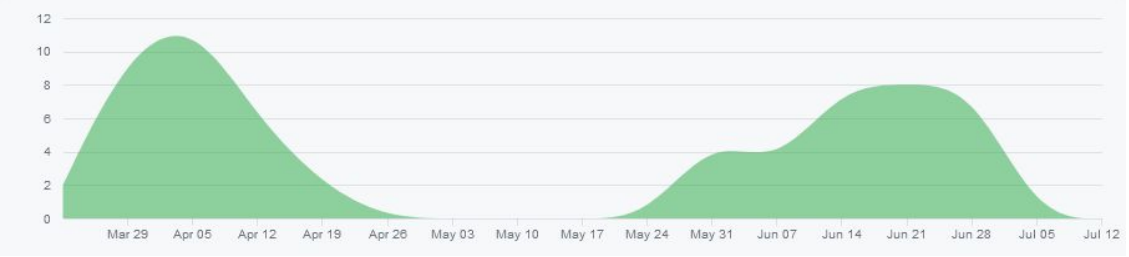


Figura 38. Gráfico de trabajo obtenido de la aplicación en GitHub

9. Conclusiones

En este capítulo se exponen las conclusiones finales del proyecto repasando la metodología de trabajo, las tecnologías utilizadas, los conocimientos adquiridos y la solución abordada para la aplicación. Por último, se comentará en extenso acerca de las mejoras y línea futura de la aplicación y lo que ha supuesto el grado en la realización de este proyecto.

9.1. Tecnologías y metodologías utilizadas

Durante la realización de este proyecto, cómo se comentó en el capítulo anterior, se tomaron ciertas herramientas de la metodología Scrum para planificar e ir gestionando nuestro proyecto. Esta fue una buena decisión debido a que a lo largo del desarrollo la ayuda que nos dio fue muy significante por los desvíos e inconvenientes que se fueron encontrando, sobre todo dadas las dimensiones finales que adquirió la aplicación. El **sistema de sprints** de dos semanas cada uno nos permitió desgranar y balancear el trabajo total, mientras al mismo tiempo nos otorgaba una visión global del proyecto y una retroalimentación continua durante el desarrollo medida también con el **control diario de dedicaciones.** Un proyecto de esta magnitud nunca se pudo desarrollar durante el grado y la elección de estos recursos nos hicieron apreciar de forma parcial la utilidad de dicha metodología de trabajo a pesar de que el desarrollo de la aplicación fuera individual.

Por otra parte, otro de los puntos más acertados para planificar el desarrollo de la aplicación fue utilizar el **Proceso Unificado de Desarrollo de Software (PUDS)**, lo que nos permitió centrarnos en la construcción de la misma mediante casos de uso y se ajustaba bien a los recursos de Scrum que habíamos tomado al estar basado en lo conocido como desarrollo incremental e iterativo. Además, a nivel conceptual, elegir este marco de desarrollo junto al **Lenguaje Unificado de Modelado (UML)** nos hizo tener una visión más clara de diseño al permitirnos visualizar y documentar las diferentes características de la aplicación.

Respecto a las tecnologías seleccionadas, su utilización ha sido una buena manera de poner en práctica todo lo aprendido durante la especialidad y conocer de forma más autodidacta tecnologías que como se dijo fueron brevemente introducidas en el grado. Django ha sido una herramienta muy potente que me ha enseñado de que son capaces los frameworks web frente a la realización de aplicaciones web a bajo nivel con PHP, JS, CSS, etc. Si se pudiera resumir esta tecnología lo haría con la palabra modularidad, cada parte en un proyecto de Django ocupa un pequeño lugar, es independiente de otras y está fuertemente relacionada con las demás. Se ha visto con el apartado de validaciones de cada modelo de datos con los formularios, su sistema propio de autentificación, la división de grupos de funcionalidades en aplicaciones, la jerarquía de las plantillas con sus librerías adicionales o hojas de estilo añadidas y con bastantes más características. Django me ha dado una forma de ver un nivel intermedio en el

desarrollo web, permitirme desarrollar gran parte de las funcionalidades en alto nivel con resultados ágiles y por otra parte poder ahondar en bajo nivel sobre soluciones que requieren más precisión.

Por otro lado MongoDB me ha supuesto apreciar las diferencias entre un diseño relacional a otro no relacional en las bases de datos, me ha permitido ver porque la utilización de este tipo de sistemas de bases de datos es cada vez más popular entre aplicaciones que necesitan una gran escabilidad de datos y un procesamiento enorme de peticiones de consulta o gestión de los mismos realizadas en paralelo. Entre sus puntos fuertes de esta tecnología me he quedado con lo cómodo que es realizar consultas de bases de datos documentales, el ahorro de recursos y la eficiencia que presenta en ciertas aplicaciones que una base de datos sea de un tipo o de otro.

Por último, tendríamos Bootstrap acompañado y soportado con JQuery, JQuery realmente aquí no ha sido una tecnología nueva a aprender pues ya se conocía esta librería de forma extensa a través de una asignatura del grado y cómo era un gran apoyo para JavaScript a la hora de manipular componentes, modificar atributos de etiquetas por distintas referencias, gestionar peticiones y hacer más cómodo y legible programar con JS, entre otras cosas. Bootstrap en este sentido ha sido un descubrimiento enorme a la hora de desarrollar webs respecto a cómo lo hacía antes utilizando únicamente CSS, su principal ventaja se podría decir que está en cómo todas sus clases están organizadas para conformar interfaces de distintas maneras y como sus componentes principales establecidos por clases siguen todos un estándar concreto de colores y formas. Sumado a esto, también está el hecho de cómo con tan poco puedes hacer webs con interfaces adaptables a dispositivos que siguen distintas resoluciones. Si antes se comentó que JQuery era un pilar para JavaScript, con Bootstrap después de su estudio se podría afirmar lo mismo para CSS.

9.2. Expectativas del proyecto y solución adoptada

Tanto en conocimientos de gestión como de ingeniería del software creo que este proyecto ha supuesto un antes y un después con lo que conocía o había realizado en el resto de asignaturas del grado.

La realización de este proyecto me ha supuesto comprender más sobre cómo organizar el trabajo de un proyecto tan grande y cómo es el desarrollo de las aplicaciones webs creadas con herramientas de alto nivel. Las condiciones a las que he estado expuesto durante el cuatrimestre me han hecho darme cuenta de lo importante que es el análisis de riesgos en un proyecto antes de ser realizado, de cómo el tiempo utilizado para desarrollarlo debe ser gestionado como un recurso más tomando en cuenta que las estimaciones pueden no ser del todo correctas al realizarlo y por otra parte cómo es familiarizarse con un proyecto tan grande que puede afrontar todo tipo de desviaciones.

Por otra parte, a nivel de software me ha hecho poner en práctica muchos de los conocimientos que se habían visto a lo largo del grado y aprender muchos otros

propios de lo conocido como un desarrollador *full stack*[12]. He aprendido gracias a introducir MongoDB el diseño de bases de datos no relacionales y sus diferencias respecto a las relacionales. He adquirido soltura programando con Python al utilizar Django y he conocido lo básico y algo más sobre los frameworks web. He elegido entre diferentes servicios de hosting cuál era el mejor para la aplicación, corrigiendo errores y comprendiendo los inconvenientes que pueden surgir cuando la despliegas en la nube. Y por último, he profundizado acerca del diseño web con Bootstrap, un apartado que en el grado no se pudo trabajar todo lo que se podría.

La idea de este proyecto ha sido muy ambiciosa y la solución adoptada creo que ha sido adecuada. **Todas las funcionalidades** que se pensaron en un principio se han **acabado implementando** y aunque no pueda dársele un uso real (debido a que no cumple la ley de protección de datos), se ha podido comprobar **toda su funcionalidad** tanto **en local** como **en la nube** realizando diferentes tipos de pruebas. Los usuarios pueden registrarse y loguearse, comunicarse, registrar o consultar artículos multimedia, compartirlos.... Ha habido grandes expectativas para el proyecto y algunas de ellas pueden no haberse cumplido, pero lo aprendido y desarrollado cumple con lo que se quería en un principio. La razón de este trabajo de fin de grado era buscar una idea propia, desarrollarla y aprender tecnologías de las que no había profundizado durante el grado por lo que los resultados obtenidos son muy satisfacibles.

9.3. Mejoras y futuro de la aplicación

El desarrollo de la aplicación ha contado con un tiempo limitado y ha sufrido varios desvíos dando como consecuencia que ciertas mejoras para la aplicación se hayan quedado como solo ideas. Gran parte son las comentadas por usuarios en el capítulo de pruebas pero siguiendo con ello hay varias más.

El apartado de notificaciones para el usuario es algo que podría mejorarse, los artículos podrían traer información sobre cuántos usuarios están interesados en ellos y las conversaciones podrían notificar los mensajes nuevos a los usuarios. Por otra parte, una lista de artículos favoritos para los usuarios también sería un gran añadido, junto a un sistema que incentivara más a los usuarios a dejar artículos en la aplicación. Una protección más sólida con la inclusión de captchas en algunas partes de la aplicación también podría ser útil, se podría aumentar aún más incluyendo un sistema de doble factor ligado al teléfono y también podría haber un apartado de reportes para notificar los usos inac. Hacer más personalizable la zona de usuarios para dejar sus redes también mejoraría la experiencia de comunicación entre ellos, incluir más idiomas para la interfaz mejoraría la accesibilidad y dejar que personalice la apariencia de su perfil de usuario también sería algo realmente bueno.

En cuanto al futuro de la misma se podría decir que es incierto, tras la realización del TFG podría considerarse su lanzamiento tras integrar los aspectos legales de la ley de

[12] Full stack developer o desarrollador full stack, viene a significar del inglés aquel desarrollador que trabaja todas las capas de una aplicación, la capa visual (front-end) y la capa lógica (back-end).

protección de datos y se podrían integrar las mejoras que se han comentado. El hosting de la misma junto a otros servicios como los que dan soporte a la base de datos de la aplicación no sería una contra a la hora de mantener puesto que su precio es relativamente asequible. Sin embargo, unas de las grandes contras que se presentan serían la gestión y vida útil de la misma debido al tiempo que supondría ambas cosas, puesto que aplicaciones de este tipo necesitan actualizaciones periódicas y por otra parte necesitan un equipo de más desarrolladores para ofrecer soporte.

La realización de este proyecto con esta aplicación como objeto del mismo ha tenido un camino complicado y al mismo tiempo ha sido una de las mejores experiencias que he tenido dentro del grado, ver como aplicar en valor práctico gran cantidad de conocimiento visto en varias asignaturas cursadas a lo largo de los años realmente ha sido enriquecedor y aunque esta aplicación pueda no llegar a su fase de producción, quedará patente aquí y se guardará un espacio en mi portafolio como futuro ingeniero.

Bibliografía

[1] Framework Web Django (Python)
https://developer.mozilla.org/es/docs/Learn/Server-side/Django

[2] Tutorial de Django Girls
https://tutorial.djangogirls.org/es/django_start_project/

[3] How to Render Django Form Manually
https://simpleisbetterthancomplex.com/article/2017/08/19/how-to-render-django-form-manually.html

[4] Django documentation
https://docs.djangoproject.com/en/2.2/

[5] Leaflet documentation
https://leafletjs.com/reference-1.6.0.html

[6] GeoPy documentation
https://geopy.readthedocs.io/en/stable/

[7] MongoDB
https://www.mongodb.com/

[8] Djongo documentation
https://nesdis.github.io/djongo/integrating-django-with-mongodb/

[9] Data Modeling Guidelines for NoSQL JSON Document Databases
https://mapr.com/blog/data-modeling-guidelines-nosql-json-document-databases/

[9] Bootstrap documentation
https://getbootstrap.com/docs/4.5/getting-started/introduction/

[10] Bootsnipp
https://bootsnipp.com/

[11] Django MathFilters documentation
https://pypi.org/project/django-mathfilters/

[12] Django Widget Tweaks documentation
https://pypi.org/project/django-widget-tweaks/

[13] StackOverflow
https://stackoverflow.com/

[14] PythonAnywhere
https://www.pythonanywhere.com/

Anexo A. Fichero de configuración global del proyecto

```
"""
Django settings for MediaKaan project.

Generated by 'django-admin startproject' using Django 3.0.3.

For more information on this file, see
https://docs.djangoproject.com/en/3.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.0/ref/settings/
"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Directorios adicionales
TEMPLATE_DIR = os.path.join(BASE_DIR,'templates')
STATIC_DIR = os.path.join(BASE_DIR,'static')
MEDIA_DIR = os.path.join(BASE_DIR,'media')

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '<SECRET KEY>'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
       'django.contrib.admin',
       'django.contrib.auth',
       'django.contrib.contenttypes',
       'django.contrib.sessions',
```

```
        'django.contrib.messages',
        'django.contrib.staticfiles',
        'widget_tweaks',
        'mathfilters',
        'gestionUsuarios',
        'gestionArticulos',
        'gestionMensajeria'
]

MIDDLEWARE = [
        'django.middleware.security.SecurityMiddleware',
        'django.contrib.sessions.middleware.SessionMiddleware',
        'django.middleware.common.CommonMiddleware',
        'django.middleware.csrf.CsrfViewMiddleware',
        'django.contrib.auth.middleware.AuthenticationMiddleware',
        'django.contrib.messages.middleware.MessageMiddleware',
        'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'MediaKaan.urls'

TEMPLATES = [
        {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [TEMPLATE_DIR,],
        'APP_DIRS': True,
        'OPTIONS': {
                'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                ],
        },
        },
]

WSGI_APPLICATION = 'MediaKaan.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/#databases
# Configuración de la base de datos que utilizaremos, en nuestro caso una
con MongoDB con el motor djongo para Django
DATABASES = {
        'default': {
        'ENGINE': 'djongo',
        'NAME': 'mediakaanadb',
         'CLIENT': {
```

```
            'host':
'mongodb+srv://user:pass@mediakaanadb-mevxh.gcp.mongodb.net/test?retryWrite
s=true&w=majority',
            'port': 27017,
            'username': '<user>',
            'password': '<pass>',
        }
    }
}


# Password validation
#
https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validator
s

AUTH_PASSWORD_VALIDATORS = [
      {
      'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
      },
      {
      'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
      },
      {
      'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
      },
      {
      'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
      },
]


# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'es-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True
```

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'
STATICFILES_DIRS = [STATIC_DIR,]

MEDIA_ROOT = MEDIA_DIR
MEDIA_URL = '/media/'

LOGIN_URL = '/gestionUsuarios/iniciosesion/'

# Para informar a través de los email
EMAIL_BACKEND='django.core.mail.backends.smtp.EmailBackend'
EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = ''
EMAIL_HOST_PASSWORD = ''
EMAIL_PORT = 587
```

Anexo B. Urls de las vistas de cada aplicación del proyecto Django

MediaKaan/urls.py ***(generales)***

```
"""MediaKaan URL Configuration

The `urlpatterns` list routes URLs to views. For more information please
see:
	https://docs.djangoproject.com/en/3.0/topics/http/urls/
Examples:
Function views
	1. Add an import:  from my_app import views
	2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
	1. Add an import:  from other_app.views import Home
	2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
	1. Import the include() function: from django.urls import include,
path
	2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.conf import settings
from django.conf.urls.static import static
from django.contrib import admin
from django.urls import path
from django.conf.urls import url, include
from gestionUsuarios import views

urlpatterns = [
	path('admin/', admin.site.urls),
	url(r'^$', views.index, name='index'),
	url(r'^about/$', views.about, name='about'),
	url(r'^gestionUsuarios/', include('gestionUsuarios.urls')),
	url(r'^gestionArticulos/', include('gestionArticulos.urls')),
	url(r'^gestionMensajeria/', include('gestionMensajeria.urls')),
	url(r'^logout/$', views.user_logout, name='logout'),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

gestionUsuarios/urls.py

```
from django.conf.urls import url
from . import views

app_name = 'gestionUsuarios'

urlpatterns = [
	url(r'^registro/$', views.register, name='register'),
```

```
      url(r'^iniciosesion/$', views.user_login, name='user_login'),

url(r'^activar/(?P<uidb64>[0-9A-Za-z_\-]+)/(?P<token>[0-9A-Za-z]{1,13}-[0-9
A-Za-z]{1,20})/$',
      views.activate, name='activate'),
      url(r'^olvidopass/$', views.forget_pass, name='forget_pass'),

url(r'^cambiopass/(?P<uidb64>[0-9A-Za-z_\-]+)/(?P<token>[0-9A-Za-z]{1,13}-[
0-9A-Za-z]{1,20})/$',
      views.change_pass, name='change_pass'),
      url(r'^miperfil/$', views.my_profile, name='my_profile'),
      url(r'^eliminarperfil/$', views.delete_user, name='delete_user'),
      url(r'^resultados/$', views.users_results, name='usrresults'),
      url(r'^usuario/$', views.user, name='usr'),
]
```

gestionArticulos/urls.py

```
from django.conf.urls import url
from . import views

app_name = 'gestionArticulos'

urlpatterns = [
      url(r'^resultados/$', views.articles_results, name='artresults'),
      url(r'^articulo/$', views.article, name='art'),
      url(r'^registrararticulo/$', views.add_article, name='add_article'),
      url(r'^misarticulos/$', views.my_articles, name='my_articles'),
      url(r'^articulosrecibidos/$', views.rec_articles,
name='rec_articles'),
]
```

gestionMensajeria/urls.py

```
from django.conf.urls import url
from . import views

app_name = 'gestionMensajeria'

urlpatterns = [
      url(r'^mensajes/$', views.message, name='msg'),
]
```

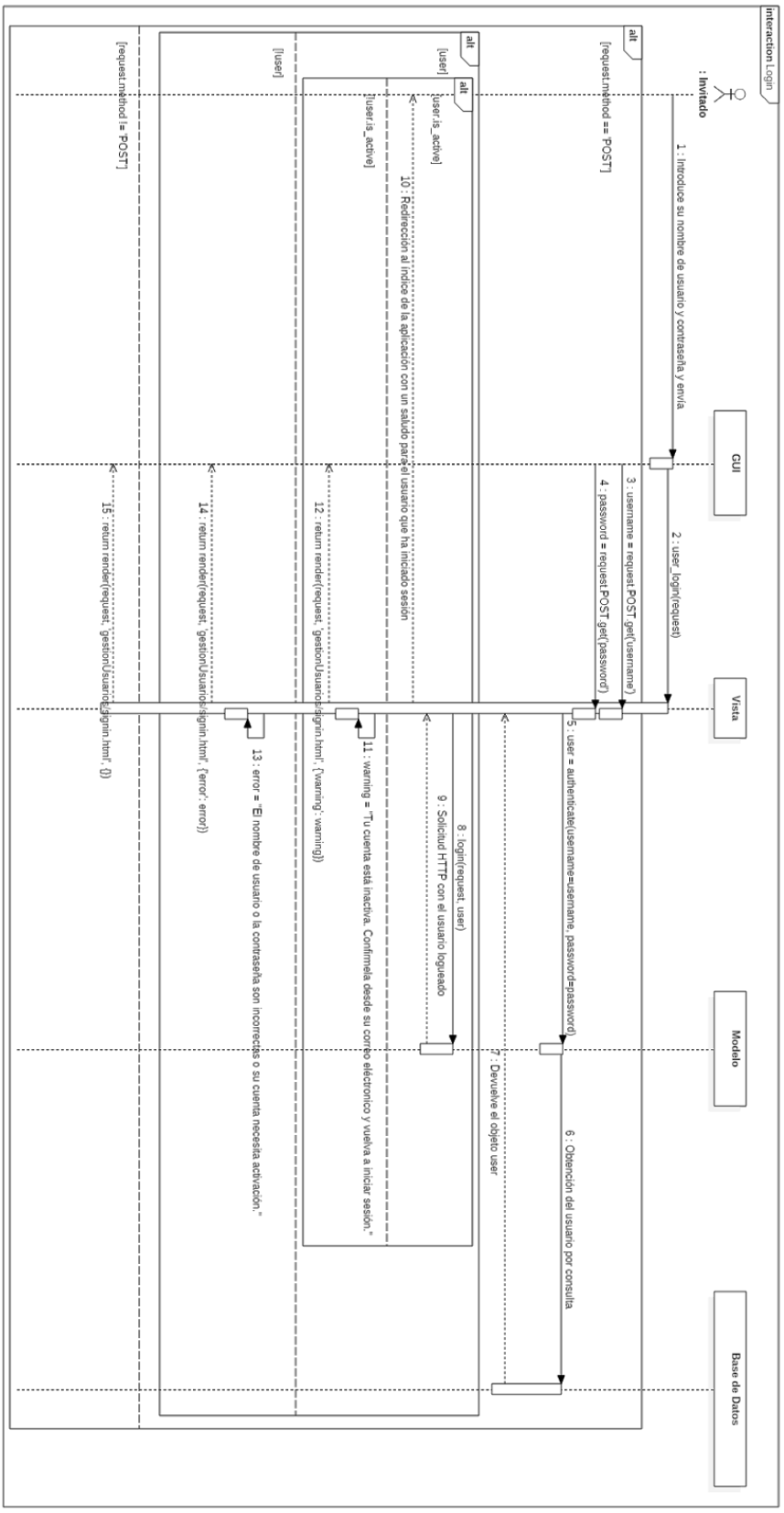
Anexo C. Imágenes generadas para la aplicación

[FOTO DE ARTÍCULO]

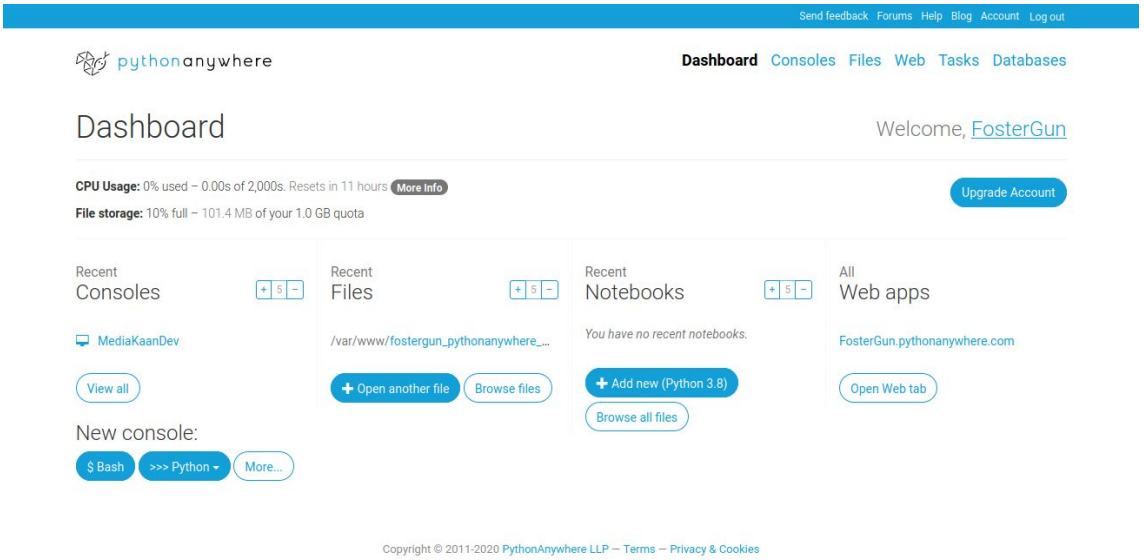
Anexo D. Borrador de horario de Junio (antes de diagrama de Gantt final)

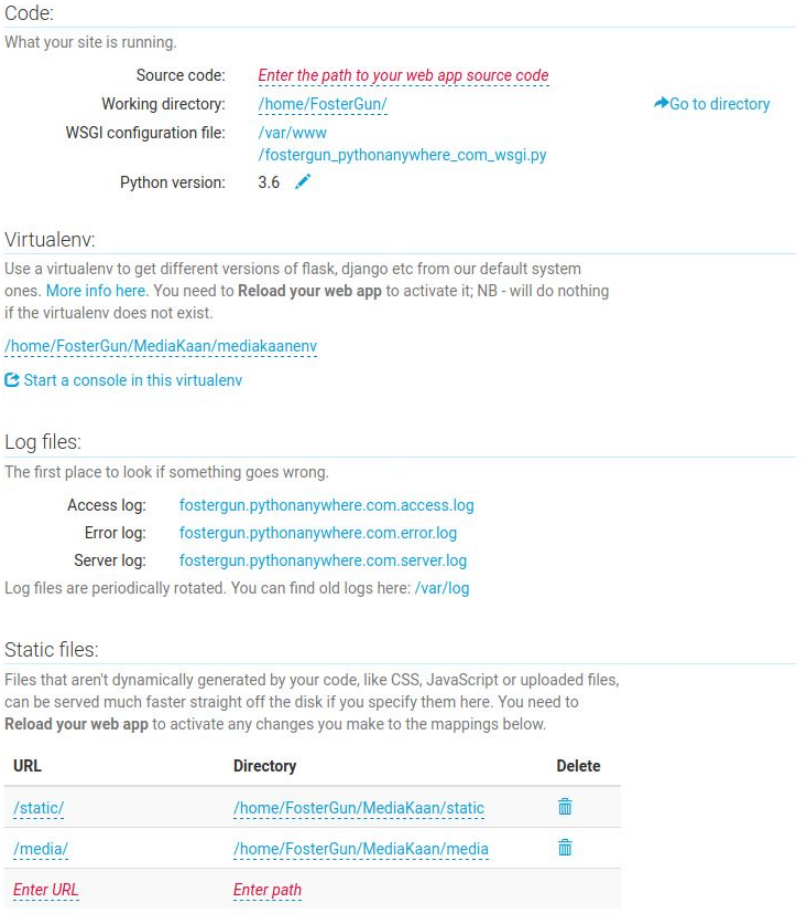
	Lunes	Martes	Miercoles	Jueves	Viernes	Sábado	Domingo
Organización	Memoria del proyecto	Diseño de la aplicación	Desarrollo de la aplicación	Desarrollo de la aplicación	Desarrollo de la aplicación	Diseño de la aplicación	Memoria del proyecto
		Desarrollo de la aplicación	Testing y depuración	Testing y depuración	Testing y depuración	Desarrollo de la aplicación	Seguimiento y control
		Testing y depuración				Testing y depuración	

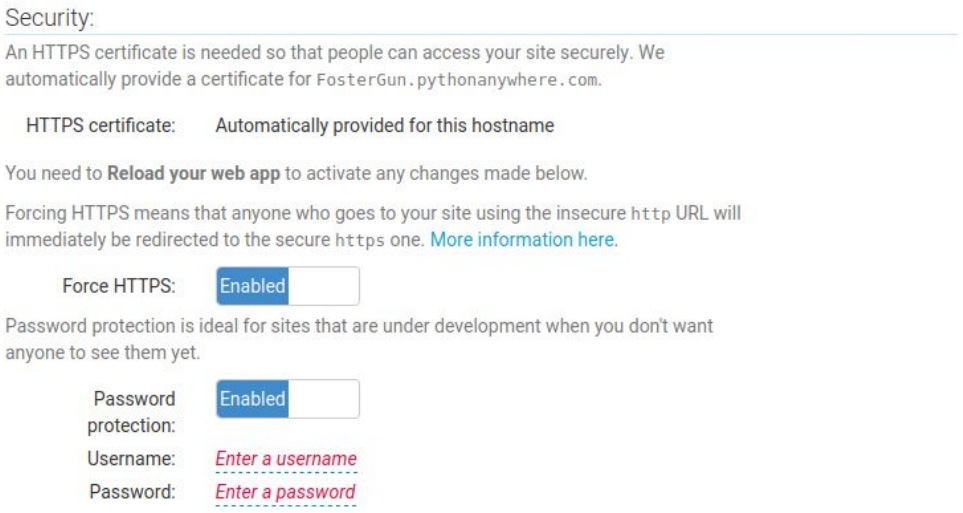
Anexo E. Diagrama de secuencia del caso de uso Iniciar Sesión



Anexo F. Configuraciones para el proyecto desplegado en la nube (PythonAnywhere)







/var/www/fostergun_pythonanywhere_com_wsgi.py

```
# This file contains the WSGI configuration required to serve up your
# web application at http://FosterGun.pythonanywhere.com/
# It works by setting the variable 'application' to a WSGI handler of some
# description.
#

# +++++++++++ DJANGO +++++++++++
# To use your own django app use code like this:
import os
import sys
#
## assuming your django settings file is at
'/home/FosterGun/mysite/mysite/settings.py'
## and your manage.py is is at '/home/FosterGun/mysite/manage.py'
path = '/home/FosterGun/MediaKaan'
if path not in sys.path:
       sys.path.append(path)

os.environ['DJANGO_SETTINGS_MODULE'] = 'MediaKaan.settings'
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```