Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DO SOFTWARE

Herramienta para la generación de blogs

Estudante: Raquel Díaz Otero
Dirección: Miguel Ángel Rodríguez Luaces
Alejandro Cortiñas Álvarez

A Coruña, setembro de 2019.

A mi familia, por haber depositado toda su confianza en mí.

Agradecimientos

A mi familia, por haber estado a mi lado cada día. Sin su apoyo y comprensión esto no habría sido posible.

A mis tutores, Alejandro y Miguel, por la ayuda y la atención que me prestaron en todo momento.

Y a mis amigos, por haber hecho esta etapa de mi vida mucho más amena.

Resumen

El objetivo de este trabajo de fin de grado es desarrollar una herramienta que permita generar el código fuente de una aplicación web para publicar y gestionar un blog a partir de una serie de características seleccionadas.

Para alcanzar este objetivo se decidió llevar a cabo el desarrollo de una línea de producto software (LPS). Para ello, se realizó un análisis del proyecto con el fin de definir los requisitos del producto y de la herramienta de generación, y para determinar la variabilidad de la LPS. Posteriormente se llevó a cabo el análisis, el diseño, la implementación y las pruebas de las funcionalidades definidas para el producto y para la herramienta de generación.

En el desarrollo del producto se empleó PostgreSQL para el almacenamiento de la información; Java, Spring e Hibernate en la implementación del servidor web; y Vue.js para la elaboración del cliente web.

En el desarrollo de la línea de producto software se empleó spl-js-engine como motor de derivación para la generación de productos, y Vue.js para la implementación de la interfaz web de la herramienta.

El trabajo de fin de grado se gestionó siguiendo una metodología iterativa e incremental, por lo que el proyecto se dividió en varias iteraciones en cada una de las cuales se llevó a cabo el desarrollo de un conjunto de funcionalidades.

Abstract

The objective of this end-of-degree project is to develop a tool that allows generating the source code of a web application to publish and manage a blog based on a group of selected features.

In order to achieve this goal, it was decided to perform the development of a software product line (SPL). For that, an analysis of the project was performed in order to define the requirements of the product and the generation tool, and to determine the variability of the SPL. Subsequently, the analysis, design, implementation and testing of the functionalities defined for the product and for the generation tool was carried out.

In the development process of the product, PostgreSQL was used for the storage of information; Java, Spring and Hibernate were used in the implementation of the web server; and Vue.js was used for the development of the web client.

In the development process of the software product line, spl-js-engine was used as a derivation engine for product generation, and Vue.js was used for the implementation of the tool's web interface.

The project was managed following an iterative and incremental methodology, so the development process was divided into several iterations in which was performed the development of a set of functionalities.

Palabras clave:

- Generación de código
- Línea de producto software
- Blog
- Servicio REST
- Vue.js
- Spring
- Hibernate
- PostgreSQL

Keywords:

- Code generation
- Software product line
- Blog
- REST service
- Vue.js
- Spring
- Hibernate
- PostgreSQL

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo se detalla la motivación que ha conducido a la realización de este proyecto y los objetivos principales definidos para este.

1.1 Motivación

Un blog es una aplicación web donde se listan cronológicamente una serie de artículos, cuyo autor puede ser siempre la misma persona o cada artículo puede tener un autor diferente. Los artículos se crean en la herramienta de administración del blog con un editor que puede ser de varios tipos y, además, el blog suele soportar una serie de funcionalidades añadidas como soporte para incluir elementos multimedia, categorización de los artículos, asociar etiquetas a los artículos, buscador de artículos, gestión de comentarios, distintos tipos de autenticación de usuarios, etc.

Las plataformas que se usan para gestionar blogs suelen ser aplicaciones complejas que incluyen un enorme conjunto de funcionalidades, independientemente de las necesidades concretas del sitio web en cuestión. Si bien son plataformas muy útiles, son aplicaciones que requieren muchos recursos y un esfuerzo considerable de administración. Además, cuando un usuario con conocimientos de desarrollo web quiere modificar algún comportamiento de la plataforma se encuentra con una tarea de mayor complejidad de la necesaria.

El objetivo principal de este trabajo de fin de grado es crear una herramienta que permita generar el código fuente de una aplicación web para publicar y gestionar un blog en base a las necesidades concretas del usuario, minimizando la complejidad de la aplicación web y de su código fuente. Con esta herramienta se pretende ofrecer una solución a los problemas mencionados, proporcionando al usuario un producto que cuente con las características deseadas y que a la vez sea lo más sencilla posible de forma que se reduzca el consumo de recursos y se evite la existencia de funcionalidades innecesarias que dificultan el uso y mantenimiento de la aplicación.

1.2 Objetivos

Para alcanzar el objetivo principal del proyecto, se deben alcanzar los siguientes objetivos específicos:

- **Desarrollar una línea de producto software** (LPS) que permita la generación de productos a partir de un conjunto de características.
- **Definir las características** que formarán parte de las aplicaciones web generadas, incluyendo las funcionalidades típicas de los blogs, como son: la gestión de artículos y usuarios, el soporte para imágenes, el uso de etiquetas y la posibilidad de añadir comentarios.
- **Analizar y modelar la variabilidad** de la línea de producto software (LPS).
- **Desarrollar el producto** de la LPS, esto es, una aplicación web para gestionar y publicar un blog que cuente con todas las características definidas.
- **Generar el código fuente** de las aplicaciones web para gestionar y publicar un blog mediante el uso de un motor de derivación de líneas de producto software.
- **Tratar el código fuente del producto** para que pueda ser empleado por el motor de derivación para generar productos.
- Proporcionar una **herramienta fácil de utilizar** mediante el desarrollo de una interfaz web que permita seleccionar las características del producto y generar las aplicaciones de forma sencilla.
- Proporcionar **productos generados de calidad**, que no tengan errores conocidos, para lo que deben realizarse tanto pruebas manuales de la aplicación como pruebas automáticas.
- Proporcionar **productos generados seguros**, que permitan el acceso a los datos solo a los usuarios autorizados, para lo que deberá emplearse algún sistema de seguridad.

Capítulo 2

Fundamentos tecnológicos

En este capítulo se describen los fundamentos tecnológicos sobre los que se ha basado el desarrollo del proyecto.

2.1 Estado del arte

En base a los objetivos definidos para el proyecto y teniendo en cuenta que el proyecto está compuesto de dos elementos principales, se ha realizado una búsqueda de herramientas existentes en el mercado que pudiesen cubrir las necesidades expresadas para cada uno de ellos.

Por una parte, se han buscado **plataformas para la gestión de blogs**, entre las cuales se han encontrado multitud de alternativas como Wordpress [5], Blogger [6], Wix [7], Medium [8], etc. Entre las principales características de estas plataformas se pueden mencionar:

- Son aplicaciones muy grandes que cuentan con un gran número de funcionalidades para gestionar un blog. Entre las funcionalidades más comunes se encuentran:
  - Gestión de artículos, contando normalmente con varios tipos de editores.
  - Gestión de usuarios, permitiendo diversos métodos de autenticación.
  - Soporte para la inclusión de elementos multimedia como imágenes y vídeos.
  - Etiquetado y categorización de artículos.
  - Gestión de comentarios, ofreciendo varios métodos a la hora de comentar.
  - Multitud de *widgets* de diversa índole como nubes de etiquetas, barras de búsqueda, *widgets* de aplicaciones externas, etc.
  - Gestión de estadísticas, permitiendo llevar un control de la actividad del blog.

- Suelen contar con gran cantidad de plantillas y contenido personalizado para modificar el aspecto del blog, y permiten extender sus funcionalidades mediante mecanismos prediseñados como plugins.
- Habitualmente tienen soporte para la suscripción de usuarios, de forma que los usuarios suscritos a un blog puedan recibir notificaciones cuando se publica un nuevo artículo.
- Están orientadas a la facilidad de uso por parte de usuarios sin conocimientos de desarrollo web, por lo que el diseño de estas páginas web es intuitivo y sencillo.
- Están diseñadas para su empleo en distintos entornos, contando con páginas web *responsive* que se adaptan a pantallas de distintos tamaños. Además, algunas de estas plataformas cuentan con aplicaciones móviles (como WordPress [5]).
- Algunas disponen de una versión gratuita y de otra versión de pago con mayor número de funcionalidades (como WordPress [9]).

El análisis de estas aplicaciones nos ha permitido observar el funcionamiento de este tipo de herramientas y ha servido de fuente de información a la hora de definir las características que formarían parte de las aplicaciones generadas con la herramienta.

Por otra parte, se ha realizado una búsqueda de **herramientas de derivación de líneas de producto software (LPS)**, entre las cuales se han encontrado las siguientes alternativas:

- **AHEAD Tool Suite (ATS)** [1]: colección de herramientas basadas en Java que ofrecen soporte para la programación orientada a características (*Feature Oriented Programming, FOP*). Estas herramientas se basan en el concepto de AHEAD (*Algebraic Hierarchical Equations for Application Design*), un modelo arquitectónico que permite representar un número arbitrario de artefactos software como conjuntos anidados de ecuaciones. El elemento principal de AHEAD es el compositor, que recibe una ecuación como entrada y la expande recursivamente hasta crear un directorio de características compuestas. En la figura 2.1 puede verse un ejemplo de composición con AHEAD. Además, los archivos de código compuestos por herramientas AHEAD están escritos en un superconjunto de Java llamado Jak (Jakarta), esto es, Java extendido con lenguajes integrados de dominio específico, máquinas de estado y metaprogramación; lo que permite que las herramientas AHEAD puedan soportar muchos dialectos de Java.
- **FeatureHOUSE** [2]: herramienta para la composición de artefactos software que es independiente del lenguaje, por lo que permite componer artefactos escritos en diferentes lenguajes de programación. En FeatureHOUSE los artefactos software se representan mediante árboles de estructura de características. La generación de los productos software se realiza mediante la combinación de los árboles de características de aquellas

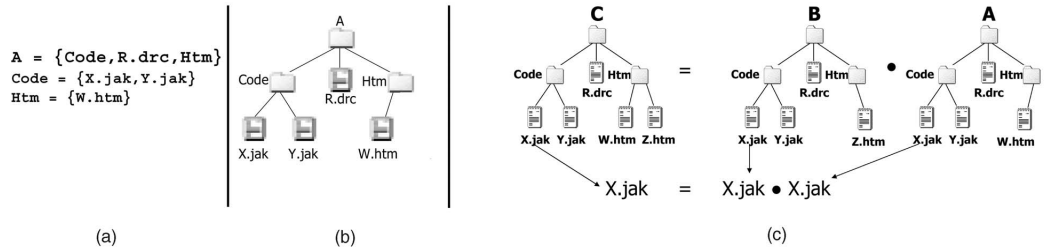


Figura 2.1: Ejemplo de composición con AHEAD [1]

funcionalidades seleccionadas para el producto. En la figura 2.2 puede verse un ejemplo de composición para un producto a partir de la combinación de árboles de características.

- **Yeoman** [3]: sistema genérico de *scaffolding* que permite la creación de aplicaciones en cualquier lenguaje de programación. Yeoman permite implementar generadores de arquitecturas de proyectos de cualquier tipo, así como utilizar alguno de sus múltiples generadores disponibles para crear directamente el esqueleto de una aplicación. Dentro del entorno de Yeoman se encuentra *yo*, la herramienta que permite la creación de proyectos utilizando plantillas de *scaffolding*, denominadas generadores. *yo* cuenta con una interfaz de línea de comandos para Node.js, como la mostrada en la figura 2.3, mediante la que se realiza la configuración y generación de los nuevos proyecto.

- **CIDE (*Colored Integrated Development Environment*)** [4]: herramienta para el desarrollo de líneas de producto software. Sigue el paradigma de separación de intereses, es decir, para la generación de productos no se extrae físicamente el código de las características, sino que se anotan fragmentos de código dentro del código original y se emplean herramientas de soporte para las vistas y la navegación. Para la anotación se utilizan colores de fondo, de modo que los fragmentos de código que pertenecen a un componente se muestran con un color de fondo. En la figura 2.4 puede verse un ejemplo de código anotado.

A la hora de emplear una herramienta de derivación de líneas de producto software en nuestro proyecto, estas opciones presentan algunos inconvenientes, como por ejemplo: son herramientas más complejas de lo necesario para el ámbito de este proyecto, algunas de ellas generan código poco legible al aplicar sus mecanismos de composición (como AHEAD o FeatureHOUSE) y además, aunque algunas permiten la composición sobre varios lenguajes de programación (como FeatureHOUSE o CIDE), en la práctica sólo funcionan con determinadas versiones del lenguaje o es necesario implementar extensiones para cada uno [10].

Para solventar estos problemas, se ha decidido utilizar en este proyecto la herramienta de

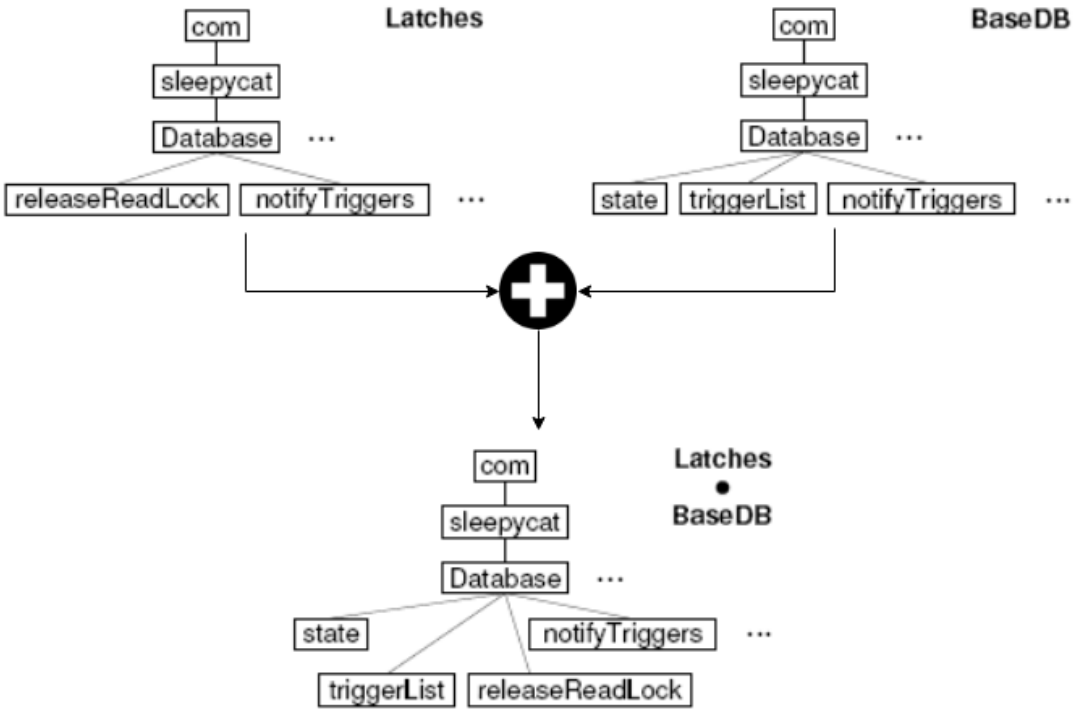


Figura 2.2: Ejemplo de composición de árboles de características con FeatureHOUSE [2]

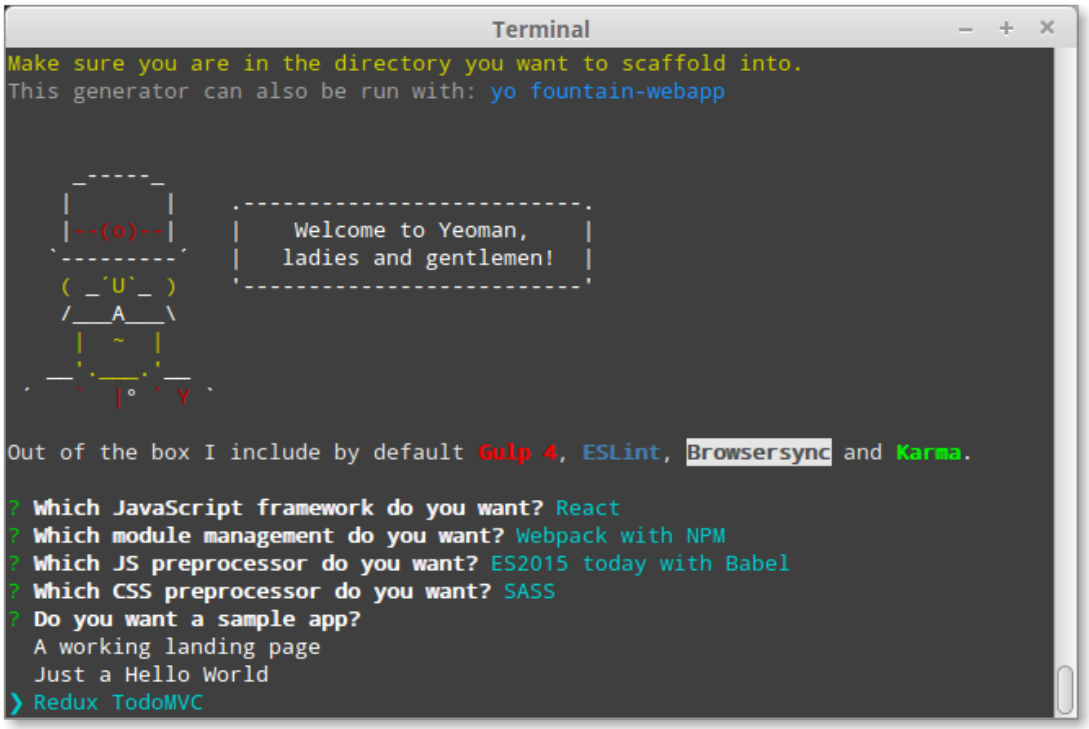


Figura 2.3: Interfaz de comandos de Yeoman [3]

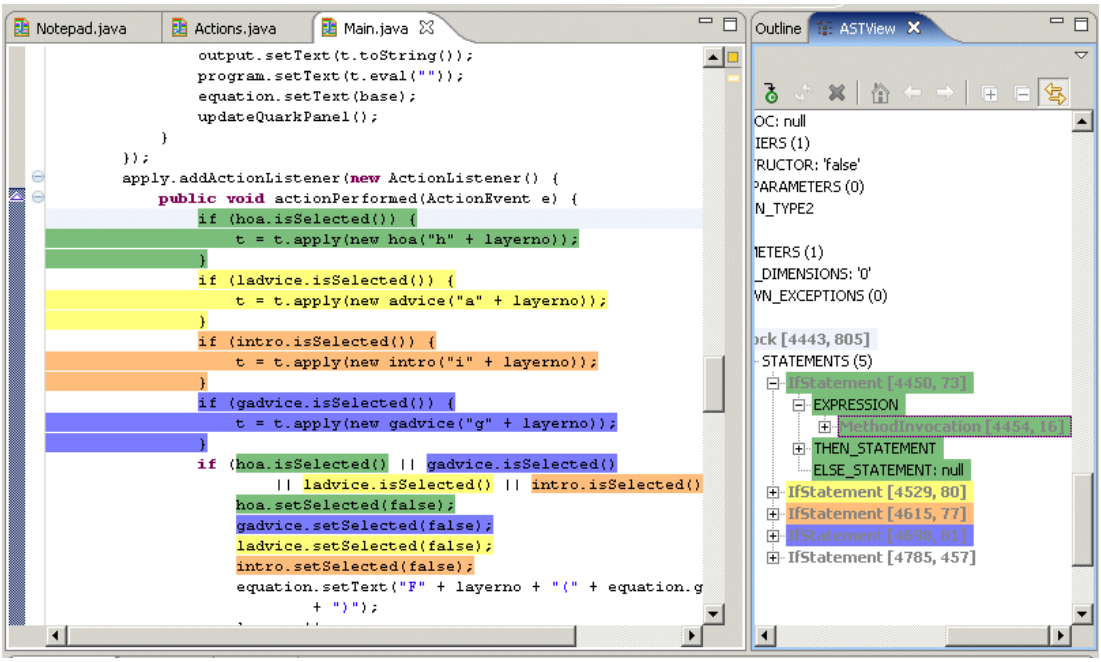


Figura 2.4: Ejemplo de código anotado con CIDE [4]

derivación spl-js-engine [11], desarrollada por uno de los directores de este trabajo para su tesis doctoral [12].

2.2 Tecnologías utilizadas

En el desarrollo de este proyecto se ha hecho uso de diversas tecnologías, entre las que se pueden destacar:

- **spl-js-engine**: herramienta JavaScript para la derivación de líneas de producto software [11].
- **Vue.js**: *framework* progresivo para construir interfaces de usuario. Su núcleo es bastante pequeño y se escala a través de plugins. Engloba en un mismo archivo HTML, CSS y JavaScript [13].
- **Node.js**: plataforma de desarrollo para la creación de aplicaciones destinadas a la Web, orientada a redes y centrada en la velocidad y la escalabilidad [14].
- **BootstrapVue**: *framework* CSS y JavaScript empleado en el desarrollo de páginas web *responsive* [15].
- **Spring**: *framework* que proporciona un modelo integral de programación y configuración para aplicaciones empresariales basadas en Java [16].

- **Hibernate**: mapeador objeto-relacional para Java que se ocupa de la persistencia de datos de la aplicación en bases de datos relacionales [17].
- **PostgreSQL**: sistema de gestión de bases de datos relacionales orientado a objetos [18].

Capítulo 3

Metodología y planificación

En este capítulo se detalla la metodología de desarrollo aplicada al proyecto, así como los mecanismos de planificación y seguimiento empleados. Además, se especifica la metodología seguida a la hora de llevar a cabo el desarrollo de la línea de producto software.

3.1 Metodología de desarrollo

Para la realización de este proyecto se ha optado por emplear una metodología de desarrollo iterativa e incremental, dirigida por las funcionalidades del sistema.

El desarrollo iterativo e incremental se basa en la división del ciclo de vida del software en varias iteraciones en secuencia, cada una de las cuales puede considerarse un pequeño proyecto autónomo compuesto por actividades de análisis, diseño, implementación y pruebas, en el que se desarrollan nuevas funcionalidades del sistema [19].

Este tipo de metodologías de desarrollo ágil se basan en la planificación adaptativa promoviendo la respuesta rápida y flexible al cambio [19], lo que proporciona una serie de ventajas frente a las metodologías de desarrollo clásicas. Las metodologías ágiles facilitan la planificación y estimación del proyecto, ya que esta se realiza para cada iteración en lugar de para el proyecto completo, y favorecen el seguimiento mediante las reuniones realizadas en cada iteración. Además, proporcionan entregas parciales del producto al cliente de forma periódica, haciéndolo partícipe del proceso de desarrollo y facilitando la detección temprana de cambios y errores. Esto, junto a la división del desarrollo en iteraciones, permite proporcionar una solución rápida a los problemas, minimizando los fallos y reduciendo los costes del producto.

La metodología de desarrollo aplicada a este proyecto está inspirada en Scrum, un marco de trabajo para el desarrollo y el mantenimiento de productos complejos, caracterizado por ser ligero y simple de entender. Scrum emplea un enfoque iterativo e incremental para optimizar la predictibilidad y el control del riesgo, y se basa en la teoría de control de procesos empírica o empirismo, que asegura que el conocimiento procede de la experiencia y permite tomar

decisiones basándose en lo conocido [20].

A continuación se describe la adaptación de Scrum aplicada al proyecto indicando qué técnicas y prácticas ha sido posible seguir teniendo en cuenta las características concretas del mismo. Para ello, se describirán los principales elementos de Scrum y su adaptación a este proyecto.

3.1.1 Equipo Scrum

Los equipos Scrum son auto-organizados y multifuncionales, y están compuestos por [20]:

- **Propietario del producto (*product owner*)**: es la persona responsable de maximizar el valor del producto y el trabajo del equipo de desarrollo. Además, es el encargado de gestionar la pila del producto, definiendo y ordenando sus elementos.
- **Equipo de desarrollo (*development team*)**: está compuesto por los profesionales encargados de crear un incremento de producto "terminado" al final de cada *sprint*.
- **Scrum Master**: es el responsable en asegurar que se entienda y se adopte Scrum.

En este proyecto el rol de propietario del producto ha sido desempeñado por los directores y la autora, ya que la definición y refinamiento de la pila del producto se ha realizado de forma conjunta entre todos los miembros del equipo, al igual que el rol de Scrum Master. El equipo de desarrollo ha estado constituido por un único miembro, la autora del proyecto.

Esta adaptación de la metodología implica varios cambios con respecto a la teoría de Scrum ya que el rol de propietario del producto ha sido desempeñado por más de una persona, al igual que el rol de Scrum Master, y el equipo de desarrollo ha contado con un único miembro en lugar de los siete que son habituales.

3.1.2 Eventos Scrum

En Scrum existen diferentes eventos predefinidos con el fin de crear regularidad y minimizar la necesidad de reuniones no definidas. Entre ellos se pueden destacar [20]:

- **Sprint**: periodo de tiempo durante el cual se crea un incremento de producto "terminado" y que constituye una iteración del ciclo de vida.
- **Scrum diario (*daily Scrum*)**: reunión que se realiza diariamente en la que el equipo de desarrollo inspecciona el trabajo realizado desde el último Scrum diario y planifica el trabajo a realizar en las siguientes 24 horas.
- **Planificación del sprint (*sprint planning*)**: reunión en la que se planifica el trabajo a realizar durante el *sprint*.

- **Revisión del sprint (*sprint review*)**: reunión que se lleva a cabo al finalizar un *sprint* con el fin de inspeccionar el incremento desarrollado y adaptar la pila del producto si fuese necesario.
- **Retrospectiva del sprint (*sprint retrospective*)**: reunión en la que se inspecciona el transcurso del *sprint* con el fin de identificar posibles mejoras sobre la forma en la que el equipo Scrum desempeña su trabajo.

En este proyecto, el ciclo de desarrollo se ha dividido en varios *sprints*, en cada uno de los cuales se llevaba a cabo el desarrollo de un conjunto de funcionalidades. Al finalizar cada *sprint* e inmediatamente antes de comenzar el siguiente se realizaba una reunión de revisión y planificación, equivalente a los eventos de planificación del *sprint* y revisión del *sprint* de Scrum. En estas reuniones, se efectuaba una revisión del trabajo realizado en el *sprint* terminado, tratando los errores y definiendo los futuros cambios a realizar. Asimismo, se llevaba a cabo el refinamiento de la pila del producto y se determinaba qué elementos de esta se desarrollarían en el *sprint* siguiente.

Puesto que el equipo de desarrollo ha estado formado por una única persona, no ha sido posible efectuar reuniones de Scrum diario, y tampoco se han realizado reuniones de retrospectiva del *sprint*.

3.1.3 Artefactos Scrum

Los artefactos de Scrum son elementos que representan el trabajo o el valor en diversas formas con el fin de proporcionar transparencia y asegurar que todos tengan el mismo entendimiento de la información clave. Los artefactos Scrum son [20]:

- **Pila del producto (*product backlog*)**: lista ordenada de todo lo que podría ser necesario en el producto. Es la única fuente de requisitos, normalmente expresados en forma de historias de usuario, y se va refinando y completando a medida que avanza el desarrollo del proyecto.
- **Pila del sprint (*sprint backlog*)**: conjunto de los elementos de la pila del producto seleccionados para llevarse a cabo en un *sprint*.
- **Incremento (increment)**: conjunto de los elementos de la pila del producto completados durante un *sprint* y de los incrementos anteriores. Al final del *sprint*, el incremento debe cumplir la definición de "terminado" y ser utilizable.

A lo largo del desarrollo de este proyecto, se han aplicado todos los artefactos definidos por Scrum. La pila del producto fue elaborada inicialmente en la fase de análisis preliminar y se fue refinando progresivamente en las reuniones de planificación y revisión realizadas.

En estas reuniones también se definió la pila del *sprint* para cada iteración que comenzaba y se realizó la revisión del incremento resultante del *sprint* finalizado. Para ello, fue necesario establecer una definición de "terminado" que asegurase el entendimiento entre los miembros del equipo a la hora de determinar que una historia de usuario había sido finalizada.

3.1.4 Herramientas de soporte a la metodología

En esta sección se mencionan las herramientas empleadas en el desarrollo del proyecto que han facilitado la aplicación de la metodología seleccionada.

Gitlab

La principal herramienta empleada ha sido una instancia de Gitlab desplegada y mantenida por el grupo de investigación al que pertenecen los directores de este trabajo, el Laboratorio de Bases de Datos. Gitlab es un repositorio de código gestionado con Git que, además, cuenta con multitud de funcionalidades que facilitan la aplicación de metodologías ágiles como: un sistema de *issue tracking*, gestión de tareas e hitos, etiquetado de *issues* y tableros para su organización, gestión de documentación mediante wikis, mecanismos para mantener la trazabilidad, mecanismos para facilitar la colaboración, etc. [21].

A la hora de trabajar con Gitlab, se crearon tres proyectos: el primero para el servidor del blog, el segundo para el cliente web del blog, y el tercero para la herramienta de generación de blogs. Todos ellos fueron asignados al mismo grupo, lo que facilitó la gestión de los hitos y de las *issues*.

Durante el desarrollo se crearon cinco hitos, cada uno de los cuales representó un *sprint*. La vista de grupo permitió gestionar de forma conjunta dichos hitos, sin necesidad de repetirlos en cada proyecto, y asignarles las *issues* de cada proyecto, proporcionando una visión global del trabajo a realizar en cada *sprint*.

A la hora de representar las historias de usuario definidas en la pila del producto, se crearon en cada proyecto las *issues* correspondientes, añadiendo un título y una breve descripción. Cada una de ellas representó tanto historias de usuario completas, como subtareas contenidas dentro de una historia de usuario. Además, también se emplearon para representar las correcciones de los *bugs* detectados durante el desarrollo y las mejoras realizadas.

Todas las *issues* se gestionaron desde la vista de grupo mediante el empleo de un tablero que facilitó su organización y planificación. Dicho tablero estaba compuesto de cuatro columnas que permitían diferenciar entre las tareas abiertas, las planificadas para su realización, las que se estaban desarrollando y aquellas que ya habían sido terminadas. Asimismo, se emplearon etiquetas para clasificar las tareas y diferenciar entre correcciones de errores, mejoras y nuevas funcionalidades.

En el momento de llevar a cabo la implementación de una *issue*, el procedimiento a seguir consistía en crear una *merge request* que producía una rama, a partir de la rama *master*, en la cual se subían los cambios realizados para la implementación y prueba de la funcionalidad en cuestión. Una vez terminada la tarea, se añadían los cambios a la rama *master* mediante la ejecución de un *merge*. Este flujo de trabajo permite diferenciar qué *commits* pertenecen al desarrollo de cada funcionalidad, favoreciendo la localización de cambios y errores.

Otras herramientas

Además de Gitlab, en la realización de las tareas del proyecto se han empleado las siguientes herramientas:

- **Eclipse** [22] y **Visual Studio Code** [23]: entornos de desarrollo integrado (*IDEs*) empleados para el desarrollo del blog y de la herramienta de generación.
- **Maven** [24]: herramienta de gestión y creación de proyectos Java, empleada en la implementación del servidor del blog.
- **JUnit** [25]: *framework* Java para la implementación y ejecución de pruebas automáticas, empleado en el proyecto para la realización de pruebas sobre el servidor del blog.
- **JaCoCo** [26]: biblioteca Java para el análisis de cobertura de código, empleada en el proyecto para analizar la cobertura de código de las pruebas automáticas sobre el servidor del blog.
- **Npm** [27]: sistema de gestión de paquetes para Node.js y entorno de ejecución para JavaScript, empleado en el desarrollo del cliente del blog y de la interfaz web de la herramienta.
- **Dbeaver** [28]: herramienta de administración de bases de datos empleada en el proyecto para facilitar el acceso a la información de la base de datos.
- **Balsamiq Mockups** [29]: herramienta para elaborar maquetas de interfaces web y de aplicaciones móviles, empleada en el proyecto en la elaboración de los prototipos de las interfaces web.
- **MagicDraw** [30] y **Draw.io** [31]: herramientas para la elaboración de diagramas.
- Navegadores web, empleados para ejecutar el proyecto y realizar pruebas manuales sobre las funcionalidades de este. Los navegadores utilizados han sido **Google Chrome** y **Firefox**.
- **Microsoft Project** [32]: herramienta de administración de proyectos, empleada para la elaboración de los diagramas de Gantt.

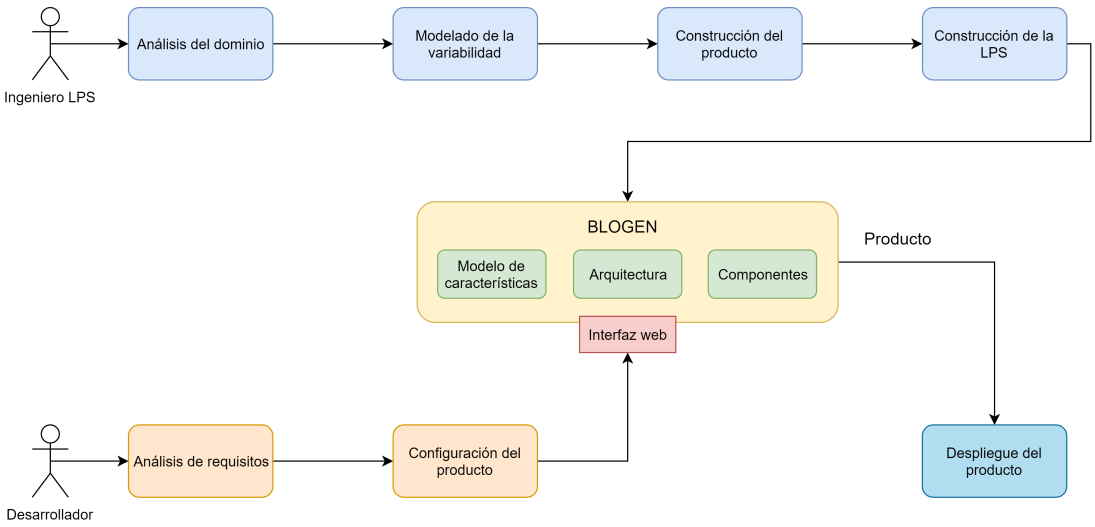


Figura 3.1: Metodología seguida para el desarrollo de la LPS

3.2 Metodología de desarrollo de la LPS

Las líneas de producto software se definen como un conjunto de sistemas software (productos), que comparten un conjunto común de características (*features*), las cuales satisfacen las necesidades específicas de un dominio o segmento particular de mercado, y que se desarrollan a partir de un sistema común de activos base (componentes reutilizables) de una manera preestablecida [33].

Para el desarrollo de la línea de producto software de este proyecto se ha empleado el motor de derivación spl-js-engine, basado en la técnica de *scaffolding*, que consiste en la generación de código a partir de plantillas predefinidas y de una especificación proporcionada por el desarrollador. Esta herramienta genera los productos a partir de plantillas, que son archivos de código fuente anotados, y una especificación de las características del producto [10]. En la sección 6.2.1 se detalla el funcionamiento de la misma.

La metodología seguida para el desarrollo de la línea de producto software está reflejada en la figura 3.1.

Como puede verse, a la hora de afrontar el desarrollo de este proyecto, se realizó un análisis del dominio de la LPS para definir las características que formarían parte de los productos generados y, en base a ellas, definir los requisitos necesarios para la construcción del producto (detallados en la sección 4.1). Además, se determinaron los requisitos de la herramienta de generación, especificados en la sección 4.3.

Una vez definidas las características, se realizó el modelado de la variabilidad. En este proceso, detallado en la sección 4.2, se determinaron las características comunes a todos los productos de la LPS, así como aquellas que podían variar de unos productos a otros. El resul-

tado de este análisis se representó mediante un modelo de características.

Tras finalizar las tareas de análisis, se llevó a cabo el desarrollo del producto, esto es, la aplicación web para publicar y gestionar un blog. Este proceso (detallado en la sección 5) estuvo guiado por el análisis previo de forma que la organización e implementación de los componentes buscó facilitar la labor de inclusión y exclusión de estos en los productos generados.

Después de la implementación y prueba del blog, se inició el desarrollo de la línea de producto software, haciendo uso de la herramienta spl-js-engine. Durante este proceso (detallado en la sección 6) se llevó a cabo la anotación del código fuente desarrollado para el blog y la implementación de la interfaz web de la herramienta.

En este punto del proceso, ya se ha completado el desarrollo de la herramienta de generación de blogs. A partir de aquí, esta podrá ser empleada para generar productos a partir de una configuración del producto, es decir, un conjunto de características seleccionadas.

3.3 Planificación y seguimiento

En esta sección se detallan los recursos necesarios para llevar a cabo el proyecto, así como la planificación y estimación realizadas inicialmente y el seguimiento real de la planificación, junto con las desviaciones en tiempo y coste.

3.3.1 Recursos

A continuación se mencionan los recursos, tanto humanos como técnicos, necesarios para llevar a cabo este proyecto.

Recursos humanos

En la realización de este proyecto se ha contado con un equipo de tres personas formado por los directores y la autora del trabajo, que han asumido los roles de *product owner*, llevando a cabo la gestión de la pila del producto; de analista, realizando actividades de planificación y diseño; y de desarrollador, efectuando la implementación y prueba de las distintas funcionalidades.

Recursos técnicos

Dentro de los recursos técnicos empleados en el desarrollo de este proyecto se pueden diferenciar dos grupos: recursos hardware y recursos software.

En cuanto a los **recursos hardware**, se ha empleado un ordenador portátil en el cual se ha llevado a cabo la implementación y prueba de todas las funcionalidades definidas para el

sistema, así como la elaboración de los diagramas y la documentación del proyecto.

Respecto a los **recursos software**, se incluyen los mencionados en la sección 3.1.4.

3.3.2 Planificación

Al comienzo del proyecto se llevó a cabo un **análisis preliminar** con el fin de determinar su alcance y las tecnologías a emplear en el mismo. La fase de análisis preliminar tuvo una duración de cuatro semanas y en ella se llevaron a cabo las siguientes tareas:

- **Estudio de la tecnología**: se realizó un análisis de los principales *frameworks* JavaScript disponibles con el fin de determinar cual era el más apropiado para desarrollar la interfaz web de la aplicación. Tras el estudio de las ventajas e inconvenientes de cada uno de ellos, se tomó la decisión de emplear Vue.js ya que es un *framework* simple y fácil de utilizar pero a la vez lo suficientemente potente para el desarrollo de este proyecto. Una vez decidido el *framework*, se procedió al estudio del mismo con el fin de obtener la formación necesaria para afrontar el desarrollo de la interfaz web del producto.

- **Definición de los actores del sistema**: se realizó un análisis de los posibles roles y actores que podrían interactuar con el sistema, definiendo las funcionalidades a las que tendría acceso cada uno de ellos.

- **Elaboración de la pila del producto inicial**: se elaboró la versión inicial de la pila del producto, definiendo las funcionalidades del sistema en forma de historias de usuario.

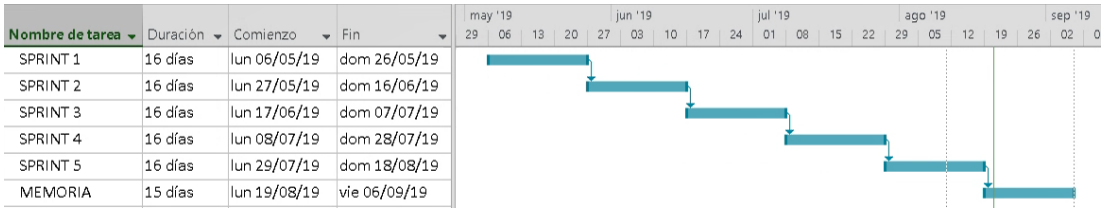
- **Elaboración del modelo de datos**: se realizó el diseño del modelo de datos para determinar como se almacenaría la información de la aplicación.

- **Elaboración de los prototipos de pantallas**: se efectuó el diseño de las pantallas de la aplicación web mediante la utilización de *mockups*.

Una vez finalizadas estas tareas se llevó a cabo la planificación del proyecto, cuyo resultado puede verse en el diagrama de Gantt de la figura 3.2. El proceso de desarrollo del software se dividió en *sprints* de tres semanas, dando lugar a cinco iteraciones tras las cuales se planificó un último *sprint* para la realización de esta memoria.

El trabajo a realizar en cada *sprint* se planificó de forma que la autora trabajaría 5 horas cada día, todos los días de la semana, en la realización de tareas de análisis, diseño, implementación y pruebas. Además, se estimó que la reunión de seguimiento y planificación correspondiente a cada *sprint* tendría una duración aproximada de una hora.

Esta planificación resulta en 105 horas de trabajo en cada *sprint* por parte de la autora, dando lugar a un total de 525 horas estimadas para el desarrollo del software. A esta cantidad



Nombre de tarea	Duración	Comienzo	Fin
SPRINT 1	16 días	lun 06/05/19	dom 26/05/19
SPRINT 2	16 días	lun 27/05/19	dom 16/06/19
SPRINT 3	16 días	lun 17/06/19	dom 07/07/19
SPRINT 4	16 días	lun 08/07/19	dom 28/07/19
SPRINT 5	16 días	lun 29/07/19	dom 18/08/19
MEMORIA	15 días	lun 19/08/19	vie 06/09/19

Figura 3.2: Diagrama de Gantt de planificación

Rol	Salario/hora
Product Owner	30 €
Analista	25 €
Desarrollador	14 €

Tabla 3.1: Salarios por hora de cada rol

se le deben sumar las 50 horas invertidas en la fase de análisis preliminar, resultando en un total de **575 horas** de trabajo estimadas para el desarrollo del proyecto.

Una vez estimado el tiempo, se realizó la estimación del coste que supondría llevar a cabo el proyecto teniendo en cuenta los salarios por hora reflejados en la tabla 3.1. Dicha estimación puede verse en la tabla 3.2.

3.3.3 Seguimiento

En esta sección se analizará el seguimiento del proyecto sobre la planificación inicial, detallando el trabajo realizado en cada *sprint* con respecto a su planificación y mencionando las desviaciones en tiempo y coste sufridas junto a las causas de las mismas. Cabe destacar que el número de historias asignadas a cada *sprint* no es el mismo, ya que no todas cuentan con la misma complejidad y el tiempo de desarrollo necesario para su implementación no es equivalente.

Hay que tener en cuenta que, siguiendo las directrices de la metodología empleada, en la reunión de revisión y planificación de cada *sprint* se realizó una revisión de las historias de usuario implementadas en la iteración finalizada, y se planificó la corrección e implementación de los errores y mejoras detectadas para el *sprint* siguiente.

Miembro	Horas *product owner*	Horas analista	Horas desarrollador	Coste
Miguel A. Rodríguez	2,5	3	-	150 €
Alejandro Cortiñas	2,5	3	-	150 €
Raquel Díaz	5	65	525	9 125 €
Totales	10	71	525	**9 425 €**

Tabla 3.2: Costes estimados para el proyecto

Sprint 1

Las tareas planificadas para el primer *sprint* fueron:

- Crear el esqueleto para el servidor web mediante el generador de Spring Boot y crear el esqueleto para el cliente web mediante el CLI de Vue.
- Implementar las operaciones de inicio y cierre de sesión (historias 1 y 2).
- Desarrollar las funcionalidades relacionadas con la gestión de artículos, esto es, la visualización, creación, modificación y borrado de artículos (historias 3, 4, 5, 6, 7, 8, 9 y 10).

Las dificultades surgidas en la configuración de Spring Security para la autenticación de usuarios y la preparación de los exámenes del segundo cuatrimestre provocaron que la duración de este *sprint* se alargase una semana, terminando el día 2 de Junio en lugar del día 26 de Mayo como estaba planificado. Además, no fue posible implementar la historia 10, por lo que tuvo que ser desarrollada en el siguiente *sprint*.

Sprint 2

Las tareas planificadas para el segundo *sprint* fueron:

- Implementar la visualización de artículos 10, asignada inicialmente al primer *sprint*.
- Desarrollar las funcionalidades relacionadas con la gestión de etiquetas, esto es, la asignación y modificación de etiquetas en artículos (historias 11, 12, 13 y 14), la visualización de etiquetas en artículos (historia 15) y la búsqueda de artículos por etiquetas (historia 16).
- Desarrollar las operaciones relacionadas con la gestión de comentarios anónimos y de usuarios registrados. Entre ellas se encuentran:
  - Visualizar comentarios (historias 17, 24 y 25).
  - Añadir y citar comentarios como usuario anónimo (historias 18 y 21).
  - Añadir, modificar, citar y eliminar comentarios como usuario registrado (historias 19, 20, 22 y 23).
  - Eliminar comentarios como administrador y como autor (historias 26 y 27).

En este *sprint* sí que fue posible llevar a cabo el desarrollo de todas las historias planificadas en tres semanas, por lo que ninguna tarea tuvo que ser aplazada.

Sprint 3

Las tareas planificadas para el tercer *sprint* fueron:

- Desarrollar las funcionalidades relacionadas con la gestión de usuarios, entre las que se encuentran:
  - Registrar un nuevo usuario (historia 28).
  - Visualizar y modificar un perfil de usuario (historias 29, 30 y 32).
  - Modificar la contraseña de un usuario (historia 31).
  - Desactivar una cuenta de usuario (historias 33 y 35).
  - Visualizar y modificar los usuarios del blog (historias 34 y 36).
  - Añadir usuarios al blog (historia 37).
- Implementar la página principal del blog (historia 38).
- Llevar a cabo el desarrollo de los *widgets* de la aplicación (historias 39, 40, 41 y 42).

En este *sprint* se cumplió la planificación, llevando a cabo el desarrollo de todas las funcionalidades previstas en un periodo de tres semanas.

Sprint 4

Las tareas planificadas para el cuarto *sprint* fueron:

- Llevar a cabo la internacionalización de los mensajes de la aplicación (historias 43 y 44).
- Implementar el *logger* para el registro de operaciones (historia 45).
- Desarrollar las funcionalidades de crear, modificar, eliminar y citar comentarios como usuario de Google (historias 46, 47, 48 y 49).
- Anotar el código fuente del blog para ser tratado por el motor de derivación de LPS.

En este *sprint* se llevaron a cabo todas las tareas planificadas en el tiempo previsto. En cuanto a la planificación, debido al retraso de una semana acarreado desde el primer *sprint* y a que se consideró que el tiempo asignado a la realización de esta memoria era insuficiente, se tomó la decisión de unir el *sprint* 5 y el asignado a la elaboración de la memoria en un último *sprint* con una duración de cinco semanas en el que se llevarían a cabo las últimas tareas de desarrollo y la redacción de la memoria.

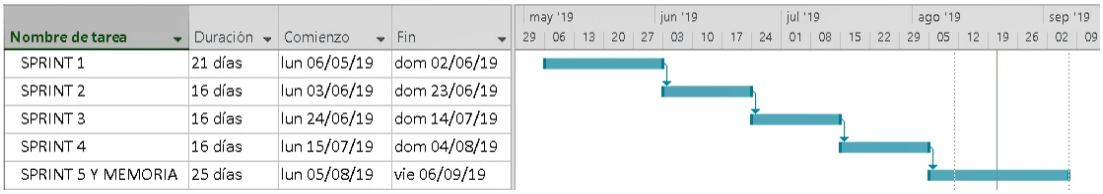


Figura 3.3: Diagrama de Gantt de seguimiento

Miembro	Horas product owner	Horas analista	Horas desarrollador	Coste
Miguel A. Rodríguez	2,5	3	-	150 €
Alejandro Cortiñas	3,5	4	-	205 €
Raquel Díaz	6	70	500	8 930 €
Totales	12	77	500	**9 285 €**

Tabla 3.3: Costes reales del proyecto

Sprint 5

Las tareas planificadas para el último *sprint* fueron:

- Desarrollar la interfaz de usuario para la herramienta de generación (historias 1, 2, 3, 4, 5 y 6 de la pila del producto de la herramienta).
- Corregir los errores detectados.
- Elaborar la memoria.

En este *sprint*, el tiempo invertido en las tareas de desarrollo fue de aproximadamente 45 horas.

Desviaciones en tiempo y coste

En la figura 3.3 se puede ver el diagrama de Gantt que refleja la progresión real del proyecto, con los cambios realizados sobre la planificación inicial y las desviaciones de tiempo sufridas. Asimismo, en la tabla 3.3 se pueden observar los costes reales del proyecto calculados en base a los salarios de la tabla 3.1 y a las horas finalmente invertidas en la realización de cada actividad.

Capítulo 4

Análisis de la línea de producto software

En este capítulo se detalla el análisis llevado a cabo para el desarrollo de la línea de producto software. En este proceso fue necesario definir los requisitos del producto y de la herramienta de generación, así como analizar y modelar la variabilidad de la LPS.

4.1 Requisitos del producto

4.1.1 Actores

En esta sección se detallan los actores definidos para el producto. Se han determinado siete actores, cuya jerarquía se puede observar en la figura 4.1.

- **Usuario**: se trata de un actor abstracto que representa las funcionalidades comunes a todos los usuarios del blog. Dichas funcionalidades son: acceder al blog y visualizar su contenido, ver los perfiles públicos de los usuarios y buscar artículos tanto por palabras clave como por etiqueta.

- **Usuario anónimo**: puede registrarse e iniciar sesión en la aplicación, publicar comentarios anónimos y citar comentarios.

- **Usuario de Google**: puede publicar, modificar, citar o eliminar comentarios mediante la autenticación con Google.

- **Usuario registrado**: puede acceder a su perfil de usuario y modificar la información asociada a su cuenta, así como publicar, citar, modificar o eliminar comentarios como usuario de la aplicación.

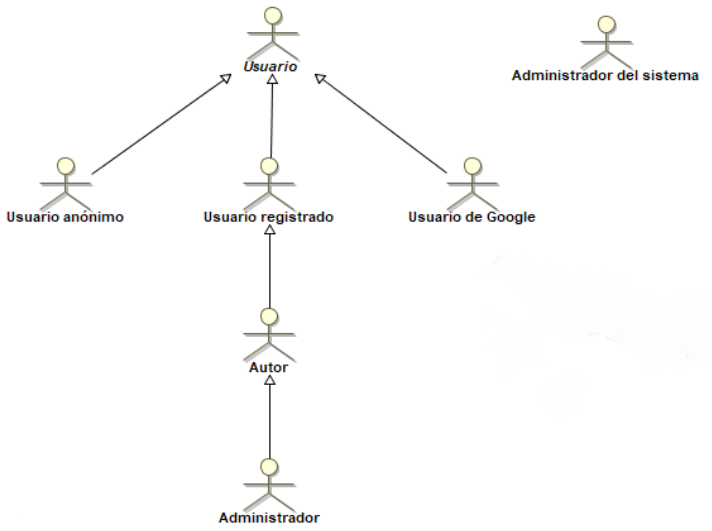


Figura 4.1: Actores

- **Autor**: tiene acceso a todas las funcionalidades del usuario registrado y además, puede editar y publicar artículos con imágenes y etiquetas, eliminar y modificar los artículos creados por él mismo, y eliminar los comentarios realizados en dichos artículos.
- **Administrador**: tiene acceso a todas las funcionalidades de la aplicación excepto a las asociadas al usuario anónimo y al usuario de Google.
- **Administrador del sistema**: tiene acceso a los archivos de *log* en los que se registran las operaciones realizadas durante la ejecución del sistema.

4.1.2 Requisitos

En esta sección se detallan los requisitos funcionales y no funcionales definidos para el producto. Inicialmente se enumeran los requisitos de forma general para posteriormente detallarlos en la pila del producto en forma de historias de usuario.

Requisitos funcionales

Los requisitos funcionales establecidos para la aplicación son los siguientes:

- Funcionalidades para la **gestión de usuarios**, permitiendo el registro y autenticación de usuarios, la visualización, modificación o borrado de una cuenta de usuario, o la creación de usuarios por parte de un administrador.

- Funcionalidades para la **gestión de artículos**, permitiendo la edición de artículos en diferentes editores de texto, así como la publicación, guardado, modificación, visualización, borrado y búsqueda de artículos.
- Funcionalidades para la **gestión de imágenes**, proporcionando varios métodos para añadir imágenes.
- Funcionalidades para la **gestión de comentarios**, permitiendo la publicación de comentarios mediante diferentes métodos, así como su modificación, borrado, cita y visualización.
- Funcionalidades para la **gestión de etiquetas**, proporcionando mecanismos para su asociación a los artículos.
- **Internacionalización** de los mensajes de la aplicación.
- Creación de un ***log*** que efectúe el registro de las operaciones realizadas.

Requisitos no funcionales

Los requisitos no funcionales definidos para la aplicación son los siguientes:

- **Facilidad de uso**: la aplicación web debe ser fácil de usar, para lo que se debe desarrollar una interfaz web intuitiva y sencilla que facilite la realización de operaciones.
- **Facilidad de instalación**: el sistema debe ser fácil de instalar y configurar, por lo que se debe proporcionar documentación con las instrucciones adecuadas.
- **Seguridad**: la aplicación debe asegurar que los datos sean accesibles únicamente por los usuarios autorizados, controlando el acceso a los mismos mediante algún sistema de seguridad.
- **Compatibilidad**: la aplicación web debe visualizarse correctamente en los navegadores Google Chrome y Firefox.

Pila del producto

Siguiendo la metodología seleccionada, se elaboró una pila del producto que constituyó la única fuente de requisitos durante el desarrollo de la aplicación. Los requisitos generales mencionados previamente se detallaron en forma de historias de usuario que, con el avance del desarrollo, se fueron refinando hasta obtener un nivel de detalle suficiente para su implementación. Asimismo, se fue reordenando la pila del producto en base a las prioridades de cada momento, hasta dar lugar a la versión final que se muestra a continuación.

1. Como usuario anónimo quiero iniciar sesión en el blog mediante la introducción de mi nombre de usuario y contraseña. Deberá ofrecerse la opción de recordar las credenciales y mostrarse un mensaje de error si los datos de acceso son incorrectos.

2. Como usuario registrado quiero poder cerrar sesión.

3. Como autor quiero acceder al editor del blog para añadir el contenido de un nuevo artículo. Dicho editor podrá ser un editor HTML, un editor Markdown o un editor WYSIWIG. El editor deberá permitir añadir enlaces al artículo mediante una ventana en la que se tendrá que indicar la url y opcionalmente el nombre del enlace. Asimismo, el editor deberá permitir añadir imágenes al artículo mediante una ventana que ofrecerá tres opciones para seleccionar imágenes:

   - Añadir una nueva imagen mediante la subida de un archivo desde el equipo. Podrá añadirse un título a la imagen.
   - Añadir una nueva imagen mediante la introducción de una url. Podrá añadirse un título a la imagen.
   - Seleccionar una o varias imágenes de la galería. Dicha galería estará formada por todas las imágenes asociadas a los artículos del blog.

   Desde el editor podrá guardarse el artículo como borrador o publicarse directamente. Además, se ofrecerá un campo de texto para introducir el título del artículo, que no será obligatorio. No se podrá guardar un artículo sin contenido.

4. Como administrador quiero visualizar todos los artículos del blog. Los artículos se mostrarán paginados y ordenados cronológicamente de forma inversa por la fecha de modificación, y para cada uno de ellos se mostrará: el título, que tendrá un enlace al artículo; la fecha de modificación; el autor, que tendrá un enlace a su perfil de usuario; el número de comentarios; las etiquetas asociadas al artículo; una opción para modificar el artículo y una opción para eliminar el artículo.

5. Como autor quiero visualizar los artículos creados por mí mismo. Los artículos se mostrarán paginados y ordenados cronológicamente de forma inversa por la fecha de modificación, y para cada uno de ellos se mostrará: el título, que tendrá un enlace al artículo; la fecha de modificación; el autor, que tendrá un enlace a su perfil de usuario; el número de comentarios; las etiquetas asociadas al artículo; una opción para modificar el artículo y una opción para eliminar el artículo.

6. Como administrador quiero acceder al editor del blog para modificar el título y el contenido de cualquier artículo del blog, tanto el texto como las imágenes y los enlaces

asociados. Si el artículo a modificar es un borrador se mostrará la opción de guardar los cambios y la opción de publicar el artículo. Si se trata de un artículo publicado únicamente se mostrará la opción de actualizar el artículo. Si el artículo había sido creado por otro usuario, se mantendrá al creador como autor del artículo.

7. Como autor quiero acceder al editor del blog para modificar el título y el contenido de un artículo creado por mí mismo, tanto el texto como las imágenes y los enlaces asociados. Si el artículo a modificar es un borrador se mostrará la opción de guardar los cambios y la opción de publicar el artículo. Si se trata de un artículo publicado únicamente se mostrará la opción de actualizar el artículo.

8. Como administrador quiero eliminar cualquier artículo del blog. Los artículos podrán eliminarse individualmente o de forma masiva y deberá mostrarse una ventana de confirmación para autorizar el borrado. Cuando se elimine un artículo deberá comprobarse si sus imágenes y etiquetas están asociadas a otros artículos, en caso contrario dichas imágenes y etiquetas deberán eliminarse.

9. Como autor quiero eliminar los artículos creados por mí mismo. Los artículos podrán eliminarse individualmente o de forma masiva y deberá mostrase una ventana de confirmación para autorizar el borrado. Cuando se elimine un artículo deberá comprobarse si sus imágenes y etiquetas están asociadas a otros artículos, en caso contrario dichas imágenes y etiquetas deberán eliminarse.

10. Como usuario quiero acceder a un artículo para visualizar su contenido. El contenido del artículo estará formado por texto e imágenes. También se mostrará la imagen y nombre de usuario del autor del artículo.

11. Como autor quiero acceder al editor del blog para añadir etiquetas a un artículo nuevo o a un artículo creado por mí mismo.

12. Como autor quiero acceder al editor del blog para modificar las etiquetas asociadas a un artículo creado por mí mismo.

13. Como administrador quiero acceder al editor del blog para añadir etiquetas a cualquier artículo del blog.

14. Como administrador quiero acceder al editor del blog para modificar las etiquetas asociadas a cualquier artículo del blog.

15. Como usuario quiero visualizar las etiquetas asociadas a un artículo. Dichas etiquetas se mostrarán como elementos independientes, y cada una de ellas tendrá un enlace que redirigirá a la búsqueda por dicha etiqueta.

16. Como usuario quiero acceder a la búsqueda por etiqueta para visualizar los artículos publicados asociados a una determinada etiqueta. Los resultados de la búsqueda se mostrarán paginados y ordenados cronológicamente de forma inversa por la fecha de publicación. Para cada artículo se expondrá su título, su fecha de publicación, un fragmento de su contenido sin imágenes, sus etiquetas (si tiene), el número de comentarios (si tiene) y un enlace que redirija al artículo completo.

17. Como usuario quiero acceder a un artículo para visualizar sus comentarios. Los comentarios se mostrarán ordenados cronológicamente de forma inversa. Para cada uno de ellos se mostrará su contenido y, además:

    - Si el autor es un usuario anónimo se mostrará un mensaje que indique que el usuario es anónimo y una imagen genérica de perfil de usuario.
    - Si el autor es un usuario registrado se mostrará su nombre de usuario y, si existe, su imagen de perfil, en caso contrario se mostrará la imagen genérica.
    - Si el autor es un usuario de Google se mostrará su dirección de correo electrónico y la imagen de perfil asociada a su cuenta de Google.

18. Como usuario anónimo quiero acceder a los comentarios de un artículo para añadir un nuevo comentario mediante la introducción del contenido en un campo de texto.

19. Como usuario registrado quiero acceder a los comentarios de un artículo para añadir un nuevo comentario mediante la introducción del contenido en un campo de texto.

20. Como usuario registrado quiero acceder a los comentarios de un artículo para modificar un comentario realizado por mí mismo. Sólo será posible modificar un comentario durante un periodo de 30 minutos desde la creación de este.

21. Como usuario anónimo quiero acceder a los comentarios de un artículo para citar un comentario a modo de respuesta. El comentario citado se mostrará en el campo de texto indicando el autor de este y su contenido entrecomillado.

22. Como usuario registrado quiero acceder a los comentarios de un artículo para citar un comentario a modo de respuesta. El comentario citado se mostrará en el campo de texto del nuevo comentario indicando el autor de este y su contenido entrecomillado.

23. Como usuario registrado quiero acceder a los comentarios de un artículo para eliminar un comentario realizado por mí mismo. Sólo será posible eliminar un comentario durante un periodo de 30 minutos desde la creación de este y deberá mostrarse una ventana de confirmación para autorizar el borrado.

24. Como administrador quiero visualizar los comentarios realizados en cualquier artículo del blog. Los comentarios se mostrarán paginados y ordenados cronológicamente de forma inversa por la fecha de creación. Para cada comentario se mostrará su contenido o un fragmento de este en el caso de que sea muy largo, la fecha de creación, el autor y una opción para eliminar el comentario.

25. Como autor quiero visualizar los comentarios realizados en los artículos creados por mí mismo. Los comentarios se mostrarán paginados y ordenados cronológicamente de forma inversa por la fecha de creación. Para cada comentario se mostrará su contenido o un fragmento de este en el caso de que sea muy largo, la fecha de creación, el autor y una opción para eliminar el comentario.

26. Como administrador quiero eliminar los comentarios realizados en cualquier artículo del blog. Los comentarios podrán eliminarse individualmente o de forma masiva.

27. Como autor quiero eliminar los comentarios realizados en los artículos creados por mí mismo. Los comentarios podrán eliminarse individualmente o de forma masiva.

28. Como usuario anónimo quiero crear una nueva cuenta en el blog. Dicha cuenta deberá estar asociada a un nombre de usuario único, una contraseña de al menos seis caracteres, un correo electrónico único y un idioma preferente. Opcionalmente se podrán indicar nombre y apellidos. Deberá mostrarse un mensaje de error si alguno de los datos introducidos no es correcto. El nuevo usuario tendrá por defecto el rol de "usuario registrado".

29. Como usuario registrado quiero acceder a mi perfil de usuario para visualizar mi imagen de perfil, mi nombre de usuario, mi nombre y apellidos, mi correo electrónico, mi idioma preferente y mi información personal.

30. Como usuario registrado quiero acceder a la modificación de mi perfil de usuario para cambiar mi imagen de perfil, mi nombre de usuario, mi nombre y apellidos, mi correo electrónico y mi información personal.

31. Como usuario registrado quiero acceder a la modificación de mi perfil de usuario para cambiar la contraseña asociada a mi cuenta. Para ello será necesario introducir la contraseña actual y la nueva contraseña en dos ocasiones.

32. Como usuario quiero acceder al perfil público de un usuario del blog para visualizar su imagen de perfil, su nombre de usuario, su nombre y apellidos, y su información personal. Si el usuario no ha actualizado su foto de perfil se mostrará la imagen por defecto.

33. Como usuario registrado quiero acceder a mi perfil de usuario para desactivar mi cuenta de usuario en el blog. Deberá mostrarse una ventana de confirmación para autorizar la acción. Una vez que una cuenta de usuario se haya desactivado no podrá activarse de nuevo.

34. Como administrador quiero visualizar todos los usuarios con permisos en el blog, incluido yo mismo. Los usuarios se mostrarán paginados y ordenados cronológicamente por la fecha de registro. Para cada uno de ellos se mostrará:su nombre de usuario, que será un enlace a su perfil de usuario; su imagen de perfil, su nombre y apellidos, su fecha de registro y su rol (administrador, autor o usuario registrado). Además, para todos los usuarios excepto el usuario autenticado se mostrará una opción para modificar el rol del usuario, y otra opción para desactivar la cuenta del usuario.

35. Como administrador quiero desactivar la cuenta de cualquier usuario asociado al blog. Una vez que una cuenta de usuario se haya desactivado no podrá activarse de nuevo.

36. Como administrador quiero modificar el rol de cualquier usuario asociado al blog, excepto el asociado a de mi cuenta. El rol de un usuario representará los permisos que posee dicho usuario en el blog.

37. Como administrador quiero añadir nuevos usuarios al blog mediante la introducción de una dirección de correo electrónico y la selección de un rol. Al confirmar los datos se creará una nueva cuenta de usuario con unas credenciales (nombre de usuario y contraseña) generadas aleatoriamente que podrán ser modificadas posteriormente, y se enviará un correo electrónico a la dirección indicada con dichas credenciales junto con las indicaciones necesarias para que el nuevo usuario pueda iniciar sesión en la aplicación.

38. Como usuario quiero acceder a la página principal del blog para visualizar su contenido. El contenido estará formado por todos los artículos publicados, que se mostrarán paginados y ordenados cronológicamente de forma inversa. Para cada uno de ellos se mostrará: el título, que será un enlace al artículo; la fecha de publicación, el contenido (texto e imágenes), las etiquetas (si tiene), el número de comentario (si tiene), el autor y la fecha de última modificación.

39. Como usuario quiero acceder a la página principal del blog para visualizar el archivo del blog, que contendrá un histórico en forma de árbol de los artículos publicados. El primer nivel mostrará el año, el segundo nivel mostrará el mes y en el tercer nivel se presentarán los títulos de los artículos publicados en el periodo correspondiente.

40. Como usuario quiero acceder a la página principal del blog para visualizar el *widget* de comentarios recientes del blog. Dicho widget mostrará los últimos 5 comentarios realizados en cualquier artículo del blog.

41. Como usuario quiero acceder a la página principal del blog para visualizar la nube de etiquetas del blog. Dicha nube de etiquetas estará formada por las etiquetas asociadas a los artículos publicados del blog y el tamaño de cada etiqueta en la nube variará dependiendo del número de artículos a los que esté asociada, siendo más grande cuanto mayor sea dicho número.

42. Como usuario quiero acceder a la búsqueda por palabras clave para visualizar los artículos publicados que contienen alguno de los términos indicados en su título o en su contenido. Los resultados de la búsqueda se mostrarán paginados y para cada artículo se expondrá su título, su fecha de publicación, un fragmento de su contenido sin imágenes, sus etiquetas (si tiene), el número de comentarios (si tiene) y un enlace que redirija al artículo completo.

43. Como usuario registrado quiero acceder a la modificación de mi perfil de usuario para establecer o modificar mi preferencia de idioma. Dicha preferencia será escogida en un selector cuyos idiomas disponibles serán: inglés, español y gallego. Los textos y fechas de la aplicación deberán mostrarse en el idioma seleccionado.

44. Como usuario quiero acceder al blog para establecer la preferencia de idioma. Dicha preferencia será escogida en un selector cuyos idiomas disponibles serán: inglés, español y gallego. Los textos y fechas de la aplicación deberán mostrarse en el idioma seleccionado.

45. Como administrador del sistema quiero acceder a los archivos del *log* del sistema para visualizar las operaciones realizadas durante su ejecución. En el *log* deben registrarse los métodos invocados junto a sus valores de entrada y salida.

46. Como usuario de Google quiero acceder a los comentarios de un artículo para añadir un nuevo comentario mediante la introducción del contenido en un campo de texto.

47. Como usuario de Google quiero acceder a los comentarios de un artículo para modificar un comentario realizado por mí mismo. Sólo será posible modificar un comentario durante un periodo de 30 minutos desde la creación de este.

48. Como usuario de Google quiero acceder a los comentarios de un artículo para eliminar un comentario realizado por mí mismo. Sólo será posible eliminar un comentario durante un periodo de 30 minutos desde la creación de este y deberá mostrarse una ventana de confirmación para autorizar el borrado.

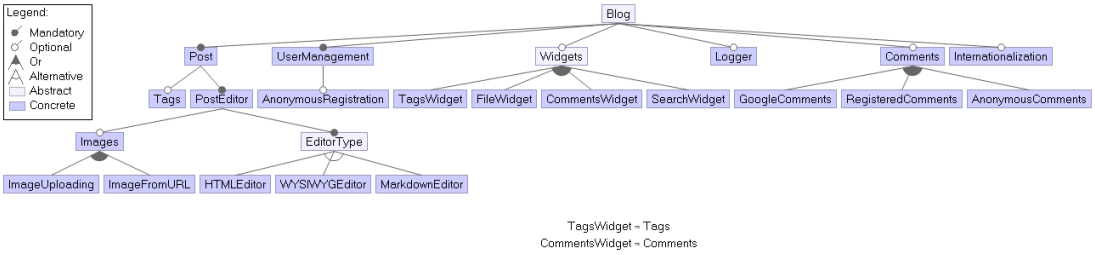


Figura 4.2: Modelo de características de la LPS

49. Como usuario de Google quiero acceder a los comentarios de un artículo para citar un comentario a modo de respuesta. El comentario citado se mostrará en el campo de texto del nuevo comentario indicando el autor de este y su contenido entrecomillado.

4.2 Modelado de la variabilidad

El desarrollo de una línea de producto software implica gestionar la variabilidad de la misma. Para ello, es necesario definir los puntos de variación entre sus productos, determinando los elementos comunes (*commonalities*) y los elementos variables (*variabilities*). Las características comunes son aquellas que están presentes en todos los productos de la línea, mientras que las características variables pueden ser comunes a varios productos, pero no a todos [34].

La variabilidad de la línea de producto software desarrollada en este proyecto se ha representado empleando un modelo de características, mostrado en la figura 4.2. Un modelo de características (*feature model*) es un árbol que estructura jerárquicamente el conjunto de funcionalidades del sistema. Dentro de esta estructura, cada característica se puede descomponer en varias sub-características que pueden ser obligatorias, opcionales o alternativas [35]. Las características obligatorias representan los elementos comunes a todos los productos (*commonalities*), mientras que las características opcionales representan los elementos variables (*variabilities*). En el anexo A se muestra la figura del modelo de características ampliada (A.1), que no ha sido posible incluir aquí por cuestiones de espacio.

Como puede verse en la figura 4.2, existen diferentes tipos de relaciones entre las características definidas para el producto en base a sus requisitos. El operador *or* indica que se tiene que seleccionar obligatoriamente alguna de las características que forman parte de la relación, pudiendo seleccionar más de una; mientras que el operador *alternative* indica que se debe seleccionar una única característica de entre las que forman parte de dicha relación.

Además, se han definido las siguientes restricciones para la selección de características: para seleccionar la característica *TagsWidget* debe seleccionarse también *Tags*, y para seleccionar la característica *CommentsWidget* debe seleccionarse *Comments*, lo que implica seleccionar al menos una de sus características hijas.

4.3 Requisitos de la herramienta de generación

En esta sección se definen los requisitos para la herramienta que permite a un desarrollador configurar un producto y luego emplear la línea de producto software para generarlo.

4.3.1 Actores

Se ha definido un único actor:

- **Usuario de la herramienta**: tiene acceso a todas las funcionalidades ofrecidas por la herramienta de generación.

4.3.2 Requisitos

En este apartado se detallan los requisitos funcionales y no funcionales definidos para la herramienta de generación, enumerándolos inicialmente de forma general para posteriormente detallarlos en la pila del producto.

Requisitos funcionales

Los requisitos funcionales definidos para la herramienta de generación son los siguientes:

- **Visualización y selección de características**: se permitirá la visualización y selección de todas las características disponibles para el producto.
- **Importación y exportación de una especificación**: se ofrecerán opciones para importar y exportar el fichero de especificación de un producto.
- **Modificación del título**: se permitirá introducir un título para el blog.
- **Generación de un producto**: podrá generarse un producto con las características seleccionadas.

Requisitos no funcionales

Los requisitos no funcionales definidos para la herramienta son los siguientes:

- **Facilidad de uso**: la herramienta debe ser fácil de usar, por lo que se debe desarrollar una interfaz web intuitiva y sencilla que facilite la generación de productos.
- **Compatibilidad**: la interfaz web de la herramienta debe visualizarse correctamente en los navegadores Google Chrome y Firefox.

Pila del producto

Los requisitos generales enumerados previamente fueron detallados en forma de historias de usuario en la pila del producto, que constituyó la fuente de requisitos durante el desarrollo de la herramienta, y que se muestra a continuación.

1. Como usuario de la herramienta quiero visualizar las características disponibles para el producto. Estas características deben presentarse en una estructura jerárquica de forma que cada funcionalidad contenga aquellas que dependan de ella y se debe verificar que se cumplan las relaciones y restricciones definidas en el modelo de características.

2. Como usuario de la herramienta quiero poder seleccionar las características que deseo que tenga el producto. Debe mostrarse una casilla de selección para cada funcionalidad que permita incluir o quitar dicha característica.

3. Como usuario de la herramienta quiero generar un producto con las características seleccionadas. El producto generado deberá comprimirse en un archivo zip y descargarse automáticamente.

4. Como usuario de la herramienta quiero poder exportar el fichero de especificación de un producto. Dicho fichero será un archivo JSON que contendrá las características seleccionadas para el producto en el momento de exportarlo.

5. Como usuario de la herramienta quiero poder importar el fichero de especificación de un producto. Dicho fichero será una archivo JSON que contendrá las características deseadas para el producto. El árbol de características deberá actualizarse con la especificación definida en el fichero importado.

6. Como usuario de la herramienta quiero añadir un título al blog que se generará. Dicho título deberá introducirse en un campo de texto y tendrá un máximo de 45 caracteres. En el caso de que no se introduzca ningún título, se pondrá uno por defecto.

Capítulo 5

Construcción del producto

5.1 Análisis

En esta sección se describen aquellos aspectos del análisis del producto no tratados en la sección 4.1.

5.1.1 Arquitectura del sistema

En este apartado se describe la arquitectura general del sistema, realizando una descomposición de alto nivel de los componentes que lo conforman. Como se puede ver en la figura 5.1, la aplicación web está basada en la arquitectura cliente-servidor, y por tanto está compuesta de dos elementos diferenciados, el servidor o *back-end* y el cliente o *front-end*. Esta arquitectura favorece la separación e independencia entre ambos elementos, facilitando que sean intercambiados o modificados sin afectar a otras partes del sistema. Además, permite la existencia de diferentes clientes para un mismo servidor y la replicación de componentes para hacer frente a la alta demanda [36].

Servidor

El servidor está formado por un conjunto de servicios que albergan la lógica de la aplicación, esto es, las operaciones de recuperación, persistencia y tratamiento de la información. Estos servicios son ofrecidos al cliente a través de una API, es decir, una interfaz que expone un conjunto de funciones (*endpoints*) que pueden ser invocadas desde el exterior.

El servidor está estructurado internamente empleando una arquitectura por capas. En este tipo de estructura, la funcionalidad del sistema está organizada en capas separadas, y cada una hace uso sólo de los servicios ofrecidos por la capa inmediatamente inferior a ella a través de una interfaz. Esta organización favorece la separación e independencia entre las capas, permitiendo su intercambio y modificación sin afectar a las demás, siempre que la interfaz no

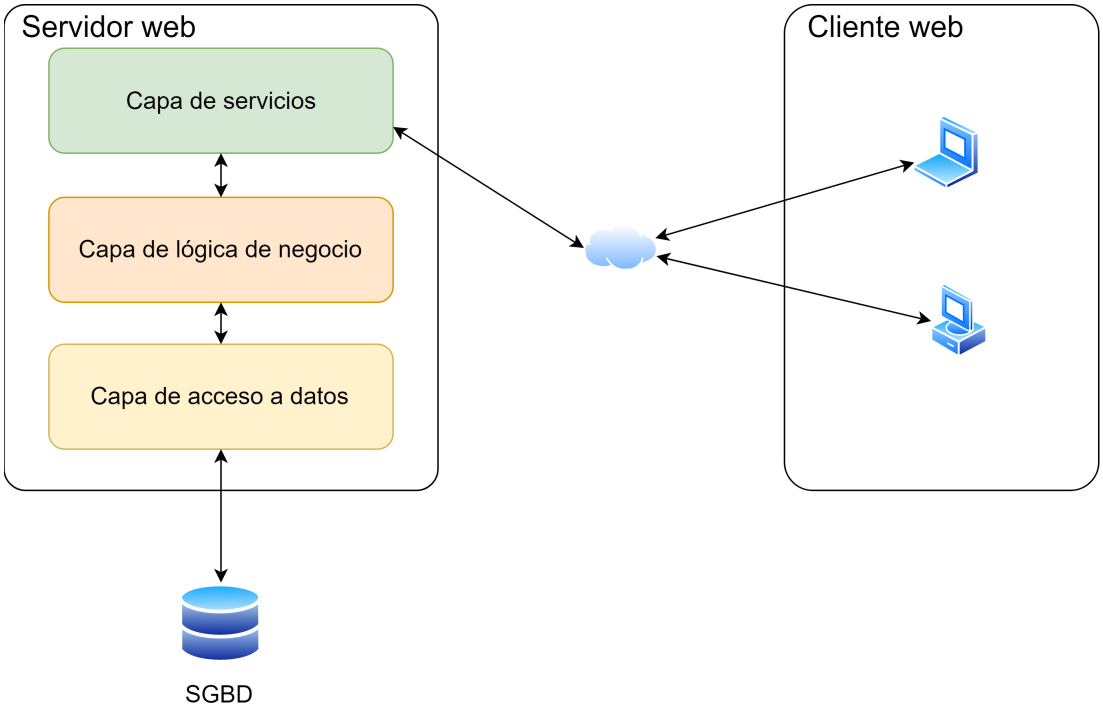


Figura 5.1: Arquitectura general del sistema

varíe [36].

Como se puede observar en la figura 5.1, el servidor está compuesto de tres capas diferenciadas, que son las siguientes:

- **Capa de acceso a datos**: es la encargada de llevar a cabo la comunicación con la base de datos, efectuando las operaciones de persistencia y recuperación de la información.

- **Capa de lógica de negocio**: efectúa las funciones principales de la aplicación realizando el procesamiento de los datos mediante la implementación de las funcionalidades disponibles.

- **Capa de servicios**: ofrece una interfaz compuesta por un conjunto de funciones que pueden ser invocadas desde el exterior. Esta capa se encarga de gestionar las peticiones recibidas del cliente invocando a los servicios de la capa de lógica de negocio, y de enviar la respuesta al cliente con los resultados obtenidos tras la ejecución de los métodos. Está basada en el paradigma REST (*Representational State Transfer*), un estilo arquitectónico que permite la comunicación con otras aplicaciones mediante el empleo de peticiones HTTP.

Cliente

El cliente es el encargado de presentar al usuario la interfaz gráfica. En base a las acciones realizadas por el usuario, el cliente efectúa las peticiones correspondientes al servidor y actualiza las vistas con la nueva información cuando recibe la respuesta.

5.1.2 Interfaz de usuario

En esta sección se pretende realizar una descripción de alto nivel de la interfaz de usuario de la aplicación, reflejando la estructura general de las pantallas más importantes que la componen, así como la navegación entre ellas.

En la fase de análisis preliminar se llevó a cabo el diseño detallado de la interfaz web de la aplicación mediante el uso de prototipos, con el fin de facilitar su futura implementación y favorecer el análisis de requisitos. Algunos de los prototipos menos importantes se muestran en el apéndice B por motivos de espacio.

La pantalla principal del blog puede verse en la figura 5.2, y cuenta con multitud de elementos e información. En la parte superior puede verse la cabecera común a todas las páginas públicas del blog cuando un usuario está autenticado. Cuando no hay usuarios autenticados, se muestra la cabecera reflejada en la figura 5.3, que permite acceder a la página de inicio de sesión y seleccionar la preferencia de idioma. Las páginas de inicio de sesión y registro pueden verse en las figuras B.1 y B.2 respectivamente.

El contenido de la página principal se divide en dos columnas. La primera de ellas muestra los artículos publicados del blog con toda su información asociada, y la segunda alberga los *widgets* de la aplicación que son: la barra de búsqueda, el archivo del blog, la nube de etiquetas y los comentarios recientes. Desde la página principal se puede navegar a multitud de vistas de la aplicación, que se describen a continuación.

Mediante la introducción de texto en la barra de búsqueda se accede a la búsqueda de artículos por palabras clave, cuyo resultado puede verse en la figura B.6. En esta vista se indica el término por el que se ha buscado y se muestran los artículos paginados.

Pulsando en las etiquetas asociadas a los artículos, se accede a la búsqueda de artículos por etiqueta, cuya página de resultados tiene un diseño prácticamente idéntico al de la búsqueda por palabras clave, y que puede verse en la figura B.7.

Pulsando en el título de un artículo se accede a su página de contenido, cuyo diseño puede verse en la figura 5.4. En esta página se muestra el título, la fecha de última actualización, el autor, las etiquetas y los comentarios del artículo. Dentro de la sección de comentarios, se encuentran los botones de opción que permiten seleccionar el tipo del comentario y un campo de texto para introducir el contenido, y a continuación se listan los comentarios realizados en el artículo. El segundo comentario listado en la figura 5.4 refleja como se mostraría la edición.

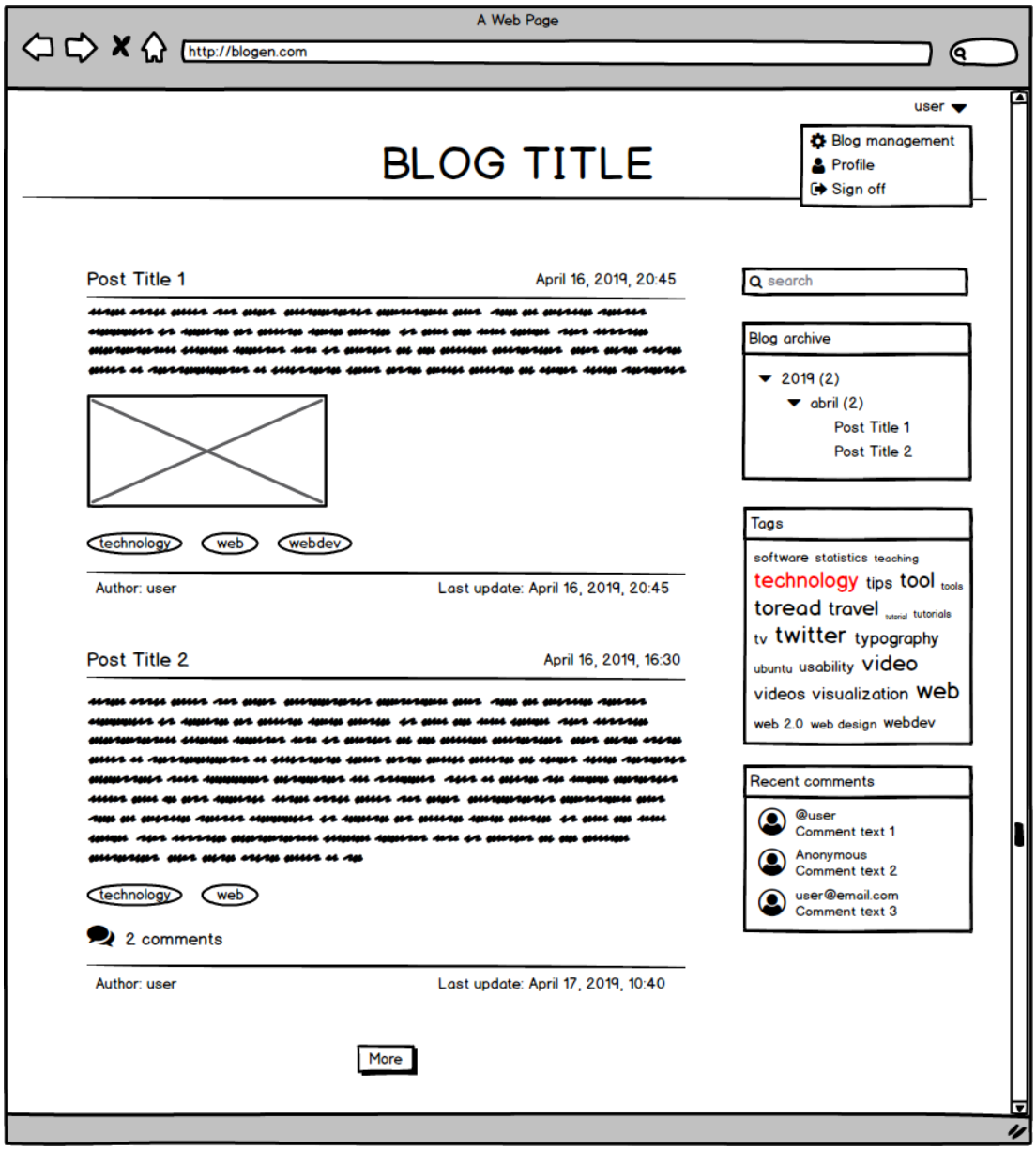


Figura 5.2: Página principal del blog

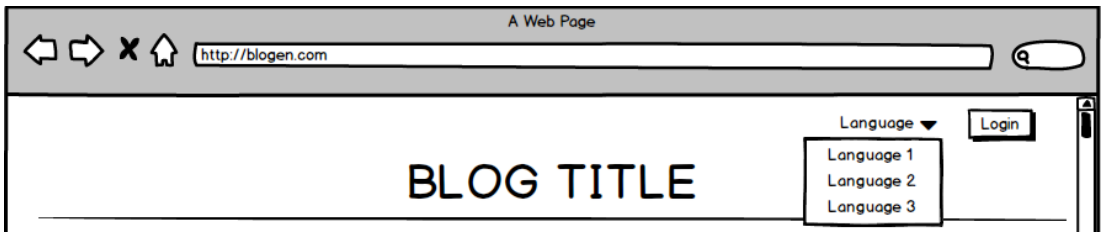


Figura 5.3: Cabecera del blog sin usuario autenticado

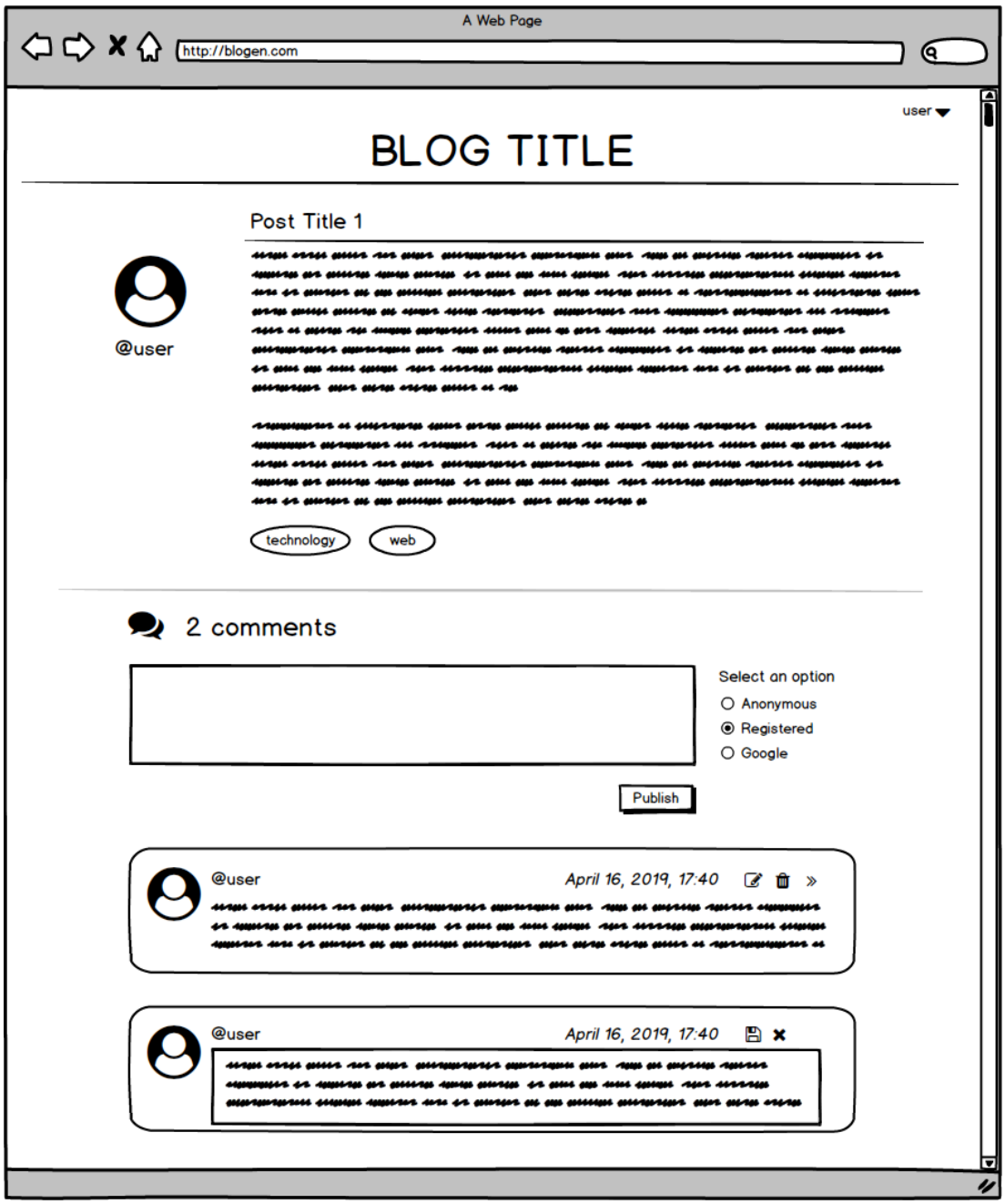


Figura 5.4: Página de un artículo

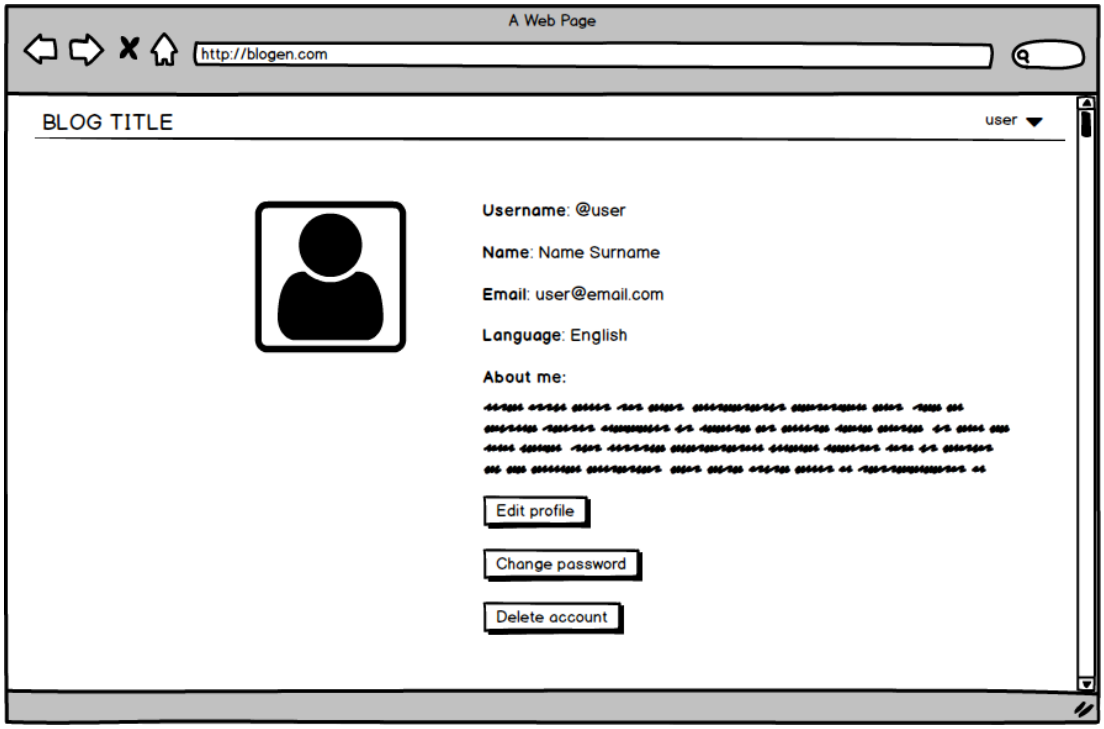


Figura 5.5: Perfil privado de un usuario

Desde la página de un artículo (figura 5.4) se puede acceder al perfil de usuario del autor pulsando en su nombre de perfil o imagen. Si el perfil pertenece al usuario autenticado, se redirigirá a su perfil privado (figura 5.5), y en caso contrario se mostrará el perfil público (figura B.9). Cabe destacar que la cabecera de las páginas privadas de la aplicación tiene un diseño diferente al de las páginas públicas, como se puede ver en la figura 5.5.

Dentro del perfil privado de un usuario se muestra la información asociada a su cuenta y se muestran tres botones que permiten acceder a la modificación del perfil de usuario, al cambio de contraseña y a la eliminación de la cuenta respectivamente. El diseño de las pantallas de la modificación del perfil y el cambio de contraseña pueden verse en las figuras B.11 y B.12 respectivamente.

A través del menú de usuario de la cabecera se puede acceder a la página de gestión del blog, que está compuesta por tres secciones con un diseño similar.

La primera de ellas es la sección de artículos, que se puede ver en la figura 5.6. En esta sección se muestran los artículos del blog con su información más relevante. Un artículo puede modificarse pulsando en el icono de editar que se muestra a su derecha, lo que provocará la redirección al editor del blog. De forma similar, el icono de la papelera permite eliminar el artículo. Para crear un nuevo artículo basta con pulsar en el botón que se muestra encima del listado, accediendo de esta forma al editor del blog.

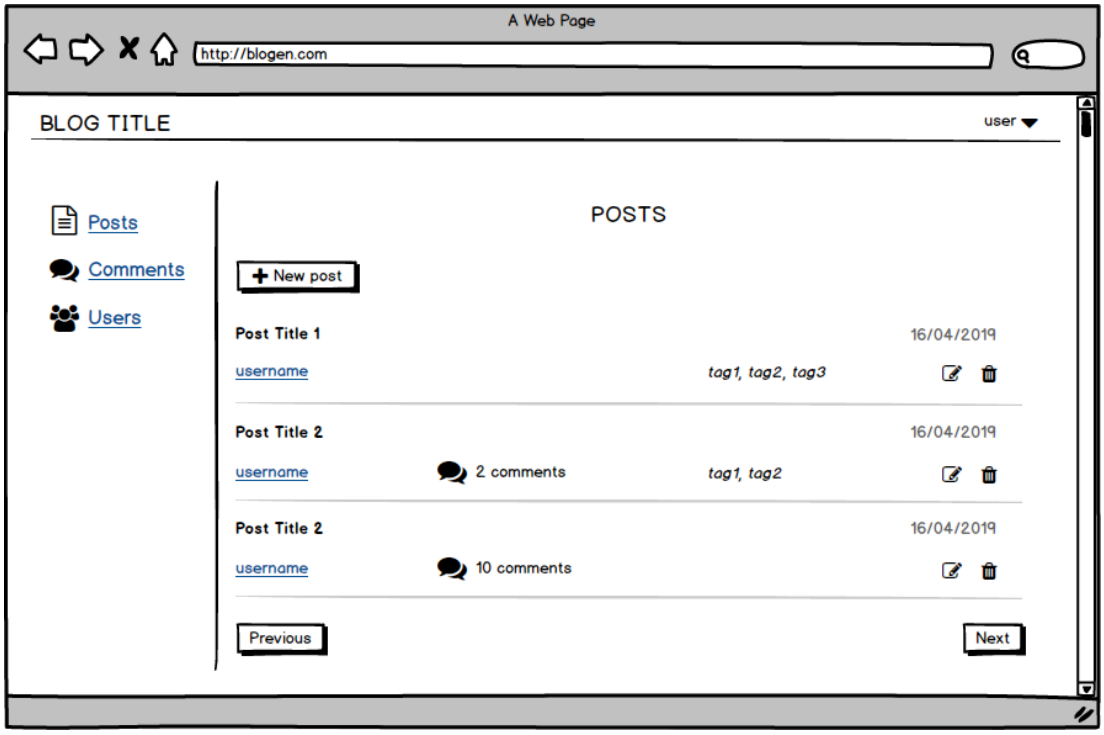


Figura 5.6: Página de gestión de artículos

La segunda sección permite la gestión de los comentarios del blog (figura B.14). En ella se listan los comentarios realizados en los artículos, mostrando su información relevante. Pulsando en el icono de la papelera situada a la derecha de cada comentario se elimina el comentario en cuestión.

El último apartado facilita la gestión de los usuarios del blog (figura B.15). Aquí se listan los usuarios registrados, mostrando su información relevante e indicando el rol de cada uno. A la derecha de cada usuario, excepto del usuario autenticado, se muestran los iconos de edición y borrado que permiten modificar el rol del usuario y eliminar su cuenta respectivamente. Además, el botón situado encima del listado permite añadir un nuevo usuario al blog (figura B.22).

Para finalizar, comentar que el blog cuenta con varios tipos de editores, todos ellos con la misma estructura. En la figura 5.7 se puede observar el diseño del editor WYSIWYG. Las páginas de los editores cuentan todas con los mismos elementos: el campo de texto para introducir el título del artículo, los botones con las acciones disponibles, el editor en cuestión y el componente para la creación de etiquetas. Dicho componente cuenta con un campo de texto en el que se introducen los nombres de las etiquetas y con un botón que permite guardar los cambios. Las figuras B.16 y B.17 muestran el diseño del editor HTML y del editor Markdown respectivamente, y las figuras B.19, B.20 y B.21 muestran las ventanas diseñadas para

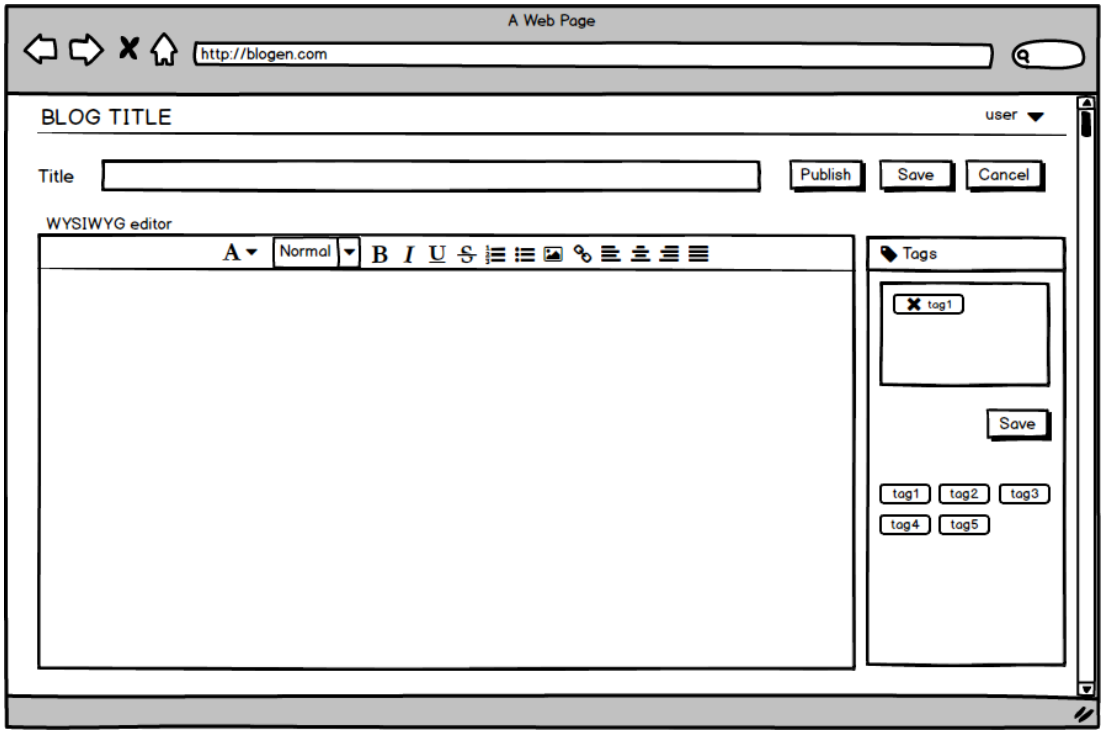


Figura 5.7: Página del editor WYSIWYG

los distintos métodos de adición de imágenes a un artículo.

La variabilidad definida para la línea de producto software influyó en el diseño de la interfaz web del producto de manera que una misma característica puede mostrarse de forma diferente dependiendo de la existencia o no de otras funcionalidades. Un ejemplo de esto es la búsqueda por palabras clave. En este caso, se diseñó una ubicación para la barra de búsqueda cuando esta característica se ha seleccionado de forma conjunta con algún otro *widget* (figura 5.2), y otra ubicación cuando el producto generado no cuenta con más *widgets* (figura B.5).

Destacar que el diseño de las pantallas aquí expuestas se ha realizado teniendo en cuenta todas las posibles características de un producto de la LPS, pero a la hora de generar un blog, el diseño de su interfaz web variará en base a las características seleccionadas para el mismo.

El diseño de todas las pantallas de la aplicación puede verse en el apéndice B.

5.1.3 Modelo conceptual de datos

En este apartado se expone el modelo de datos diseñado en la fase de análisis preliminar para determinar como se modelaría la estructura de la base de datos. Dicho modelo de datos puede verse en la figura 5.8 y de su diseño cabe destacar:

- **UserProfile** representa a un usuario de la aplicación. Esta entidad almacena los datos

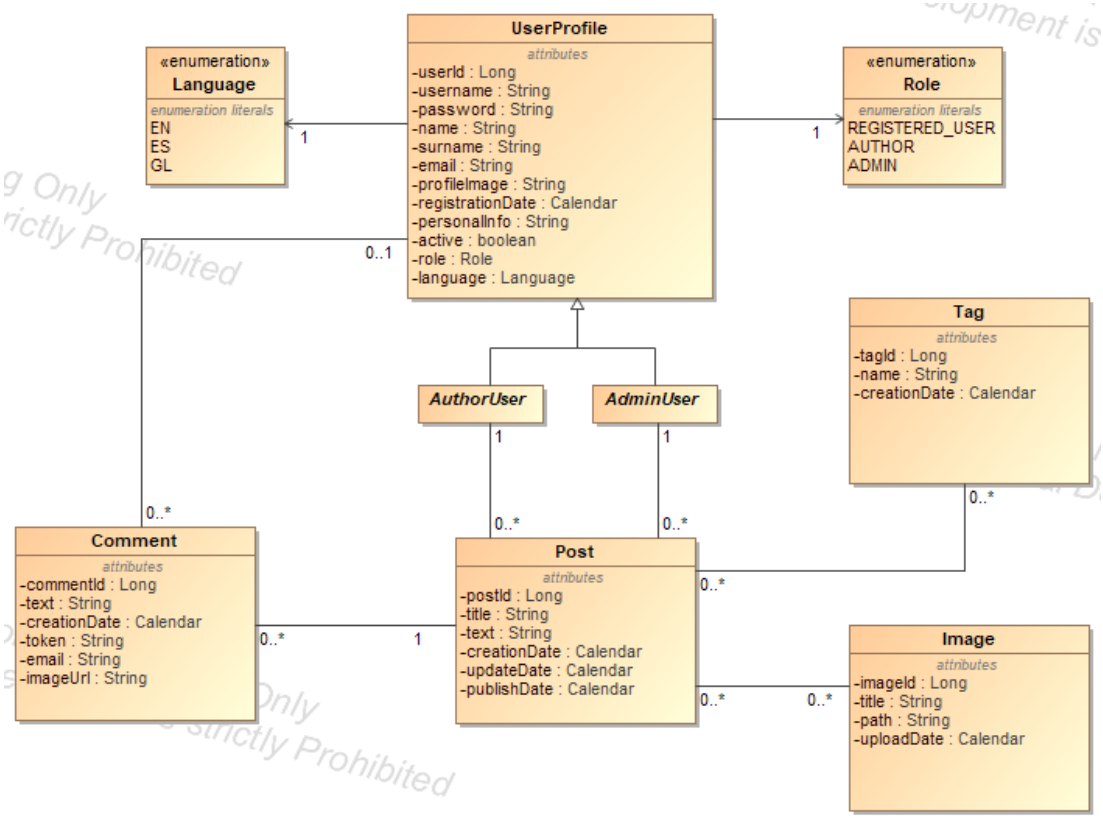


Figura 5.8: Modelo conceptual de datos

personales del usuario y, además, tiene un rol asociado que determina sus permisos en la aplicación, una preferencia de idioma y un estado ("active") que establece si la cuenta de usuario está activa o inactiva. Las clases *AuthorUser* y *AdminUser* se emplean en el modelo de datos para representar a usuarios con rol *AUTHOR* y *ADMIN* respectivamente. Un usuario con alguno de estos roles puede ser autor de ninguno o muchos artículos y cualquier usuario puede ser autor de ninguno o varios comentarios.

- **Comment** representa un comentario en un artículo del blog, por lo que la entidad comentario debe estar asociada obligatoriamente a un único artículo. Un comentario puede tener o no un usuario asociado, diferenciando así los comentarios anónimos de aquellos realizados por usuarios registrados. Además, cuenta con las propiedades *token*, *email* e *imageUrl* para almacenar los datos de un usuario de Google, ya que al no tratarse de un usuario de la aplicación no se almacena en la base de datos como una instancia de UserProfile.

- **Tag** representa una etiqueta. Las etiquetas se añaden a artículos, pero no es obligatorio que estén asociadas a un artículo.

- **Image** representa una imagen del blog. Las imágenes se añaden a artículos, pero no es obligatorio que estén asociadas a un artículo.

- **Post** representa un artículo del blog y debe estar asociado obligatoriamente a un único autor, que es el usuario que lo crea y que debe tener el rol *AUTHOR* o el rol *ADMIN*. Además, puede tener comentarios, imágenes y etiquetas asociadas.

Es importante resaltar que el modelo de datos aquí descrito está compuesto por todos los posibles elementos que pueden formar parte de un producto de la LPS, pero a la hora de generar un blog, su modelo de datos variará en función de las características que hayan sido seleccionadas para el mismo.

5.2 Diseño

5.2.1 Arquitectura tecnológica del sistema

En esta sección se pretende complementar el análisis de la arquitectura realizado en la sección 5.1.1 (figura 5.1), a través del estudio de las tecnologías empleadas en cada componente de la aplicación. La figura 5.9 ilustra la arquitectura tecnológica de la aplicación.

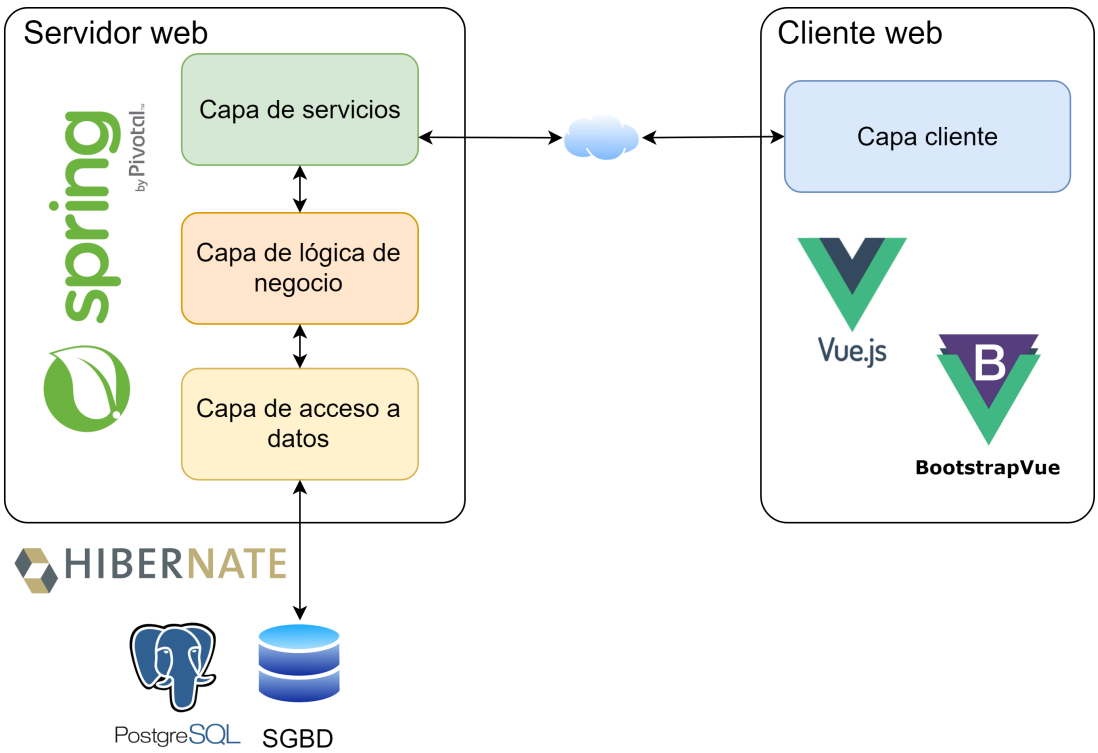


Figura 5.9: Arquitectura tecnológica del sistema

Base de datos

La tecnología empleada para gestionar el almacenamiento de la información ha sido PostgreSQL, un sistema de gestión de bases de datos relacional de código abierto. En la práctica se han creado dos bases de datos, la primera, para almacenar los datos de la aplicación generados durante su ejecución; y la segunda, para llevar a cabo la ejecución de las pruebas automáticas sobre el servicio REST.

Servidor

En el desarrollo del servidor, se ha optado por emplear Java EE 8 junto con las funcionalidades ofrecidas por el *framework* Spring. Además, se ha empleado Hibernate, un mapeador objeto-relacional que ha facilitado la persistencia y recuperación de información de la base de datos gracias a que permite trabajar con objetos anotados en lugar de hacerlo directamente con las tablas de la base de datos.

Dentro de la estructura interna del servidor, se han utilizado de los siguientes elementos:

- En la **capa de acceso a datos** se han empleado **repositorios**, que son clases que se encargan de la comunicación con el sistema de almacenamiento llevando a cabo la eje-

cución de las operaciones de recuperación y persistencia de la información. Los repositorios se basan en el patrón DAO, cuya principal ventaja es que aísla la capa de acceso a datos del resto de la aplicación favoreciendo que los cambios en el sistema de almacenamiento no afecten a otras partes del sistema. Los repositorios ofrecen sus servicios a la siguiente capa mediante una interfaz.

- En la **capa de lógica de negocio** se han empleado **servicios**, que son clases que agrupan un conjunto de funcionalidades relacionadas que son ofrecidas a la capa superior a través de una interfaz. Los servicios hacen uso de las funcionalidades expuestas por los repositorios a la hora de implementar la lógica de los métodos.

- En la **capa de servicios** se han empleado **controladores**, que son clases encargadas de recibir las peticiones del cliente y enviarle la respuesta tras haber invocado a los métodos ofrecidos por los servicios. Los controladores reciben las peticiones HTTP del cliente y envían la respuesta en formato JSON a través de una nueva petición HTTP.

Para gestionar la seguridad de la aplicación se ha hecho uso de Spring Security, un *framework* de control de acceso y autenticación empleado para securizar aplicaciones basadas en Spring [37]. Este *framework* ha permitido configurar la autenticación en la aplicación y establecer restricciones de acceso a las peticiones mediante el uso de filtros. En cuanto a la comunicación entre el cliente y el servidor, se ha empleado el estándar *JSON Web Token (JWT)* para gestionar la autorización de usuarios.

Cliente

En el lado cliente se ha hecho uso de Vue.js, un *framework* JavaScript empleado en el desarrollo de interfaces web y *Single Page Applications* (SPA) [13]; y de BootstrapVue, un *framework* CSS empleado en esta aplicación para añadir estilos a los componentes.

Para complementar las funcionalidades de Vue se han empleados las siguientes bibliotecas:

- **Vue-router**: es el router oficial de Vue para el desarrollo de *Single Page Applications* [38]. Esta biblioteca permite la navegación entre vistas pudiendo configurar interceptores para ejecutar acciones antes o después de la navegación.

- **Vuex**: es una biblioteca y patrón de gestión de estado que actúa como un repositorio central de información para todos los componentes de la aplicación [39]. Vuex permite gestionar desde un único punto la información común de numerosos componentes. Desde los componentes se pueden invocar los métodos que ofrece Vuex para mutar los datos almacenados, así como para leer sus valores.

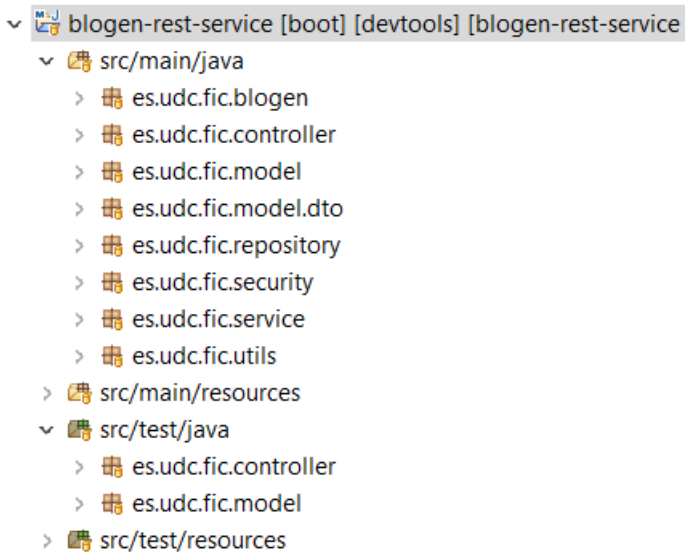


Figura 5.10: Estructura de paquetes del servidor

- **Axios**: es un cliente HTTP basado en promesas empleado en el desarrollo de aplicaciones JavaScript, que funciona tanto en navegadores como en plataformas de Node.js [40]. Axios facilita el consumo de servicios web que devuelven información en formato JSON y permite la configuración de interceptores a las peticiones, pudiendo ejecutar acciones antes de que estas sean enviadas al servidor o, al ser recibidas, antes de que lleguen al componente.

5.2.2 Diseño de la aplicación

En esta sección se detallará la estructura interna de los componentes de la aplicación.

Servidor

La estructura de paquetes del servidor se muestra en la figura 5.10. De cada paquete se puede destacar:

- **src/main/java**: contiene las clases Java que implementan las funcionalidades del servidor. Internamente está estructurado de la siguiente forma:
  - **blogen**: contiene la clase principal, que se encarga de la ejecución de la aplicación.
  - **controller**: contiene los controladores de la aplicación.
  - **model**: contiene las entidades y los DTOs que forman parte de la aplicación.
  - **repository**: contiene los repositorios (DAOs) de la aplicación.

  - **security**: contiene clases encargadas de los controles de seguridad.
  - **service**: contiene los servicios de la aplicación.
  - **utils**: contiene clases con funcionalidades comunes.
- **src/main/resources**: contiene archivos de configuración, *scripts* SQL y archivos estáticos empleados en la aplicación.
- **src/test/java**: contiene los archivos de pruebas. Internamente está subdividido en dos paquetes:
  - **controller**: contiene los archivos de pruebas de los métodos expuestos por los controladores.
  - **model**: contiene los archivos de pruebas de las entidades.
- **src/test/resources**: contiene archivos de configuración de las pruebas automáticas, *scripts* SQL y archivos estáticos empleados en las pruebas.

A continuación se detallan los elementos de los paquetes más importantes.

Entidades. Son las clases persistentes que representan las entidades diseñadas en el modelo de datos (figura 5.8). Estas clases cuentan con una serie de atributos que representan tanto las propiedades definidas para cada entidad en el modelo de datos, como las relaciones entre entidades. Además, tienen métodos de lectura y escritura (*getters* y *setters*) para permitir el acceso a sus atributos, constructores y los métodos *equals*, *hashcode* y *toString*.

Todas las entidades están anotadas con @Entity con el fin de indicar al mapeador objeto-relacional que son clases persistentes y deben ser mapeadas a la base de datos. Además, se han empleado otras anotaciones de Hibernate en los atributos para determinar cómo deben ser generadas las tablas en la base de datos.

Repositorios (DAOs). Son los elementos encargados de proporcionar métodos para realizar las operaciones de persistencia y recuperación de la información de la base de datos. Existe un repositorio para cada entidad, excepto para *Post* ya que, debido a necesidades de implementación, ha sido imprescindible crear otro repositorio a mayores (cuestión detallada en la sección 5.3.1). La estructura de repositorios de la aplicación puede verse en la figura 5.11.

Todos los repositorios están anotados con @Repository y heredan de la interfaz JpaRepository de Spring que proporciona una serie de métodos CRUD. Esta configuración permite hacer uso de las ventajas que proporciona Spring Data como, por ejemplo, la implementación automática de métodos siguiendo una convención de nombrado, la posibilidad de implementar consultas mediante el uso de la anotación @Query, o las facilidades a la hora de realizar la paginación de resultados.

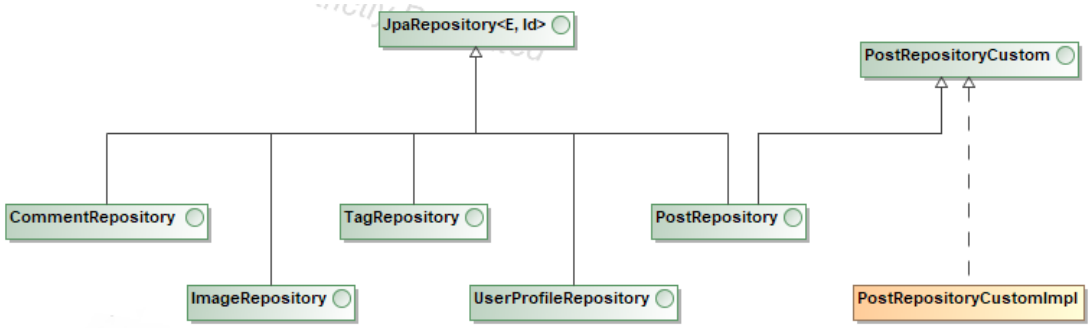


Figura 5.11: Estructura de los repositorios

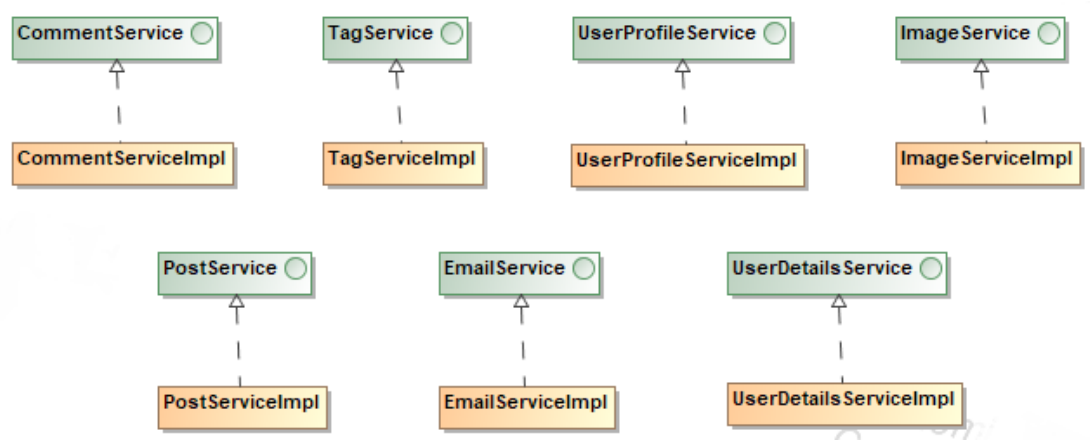


Figura 5.12: Estructura de los servicios

Servicios. Son las clases en las que se implementa la mayor parte de la lógica del servidor. Cada servicio agrupa un conjunto de funcionalidades relacionadas, y todos están anotados con @Service. Los servicios definidos pueden verse en la figura 5.12. Todos ellos cuentan con una interfaz, a través de que la que ofrecen sus métodos a los demás componentes, y con una clase que realiza la implementación de dichos métodos. Cabe destacar que la clase *UserDetailsServiceImpl* implementa una interfaz ofrecida por Spring (*UserDetailsService*) y es utilizada en el proceso de autenticación de usuarios.

Los servicios, mediante inyección de dependencias, hacen uso de los métodos expuestos por los repositorios y por otros servicios a través de sus interfaces.

Controladores. Son las clases encargadas de exponer los servicios al exterior a través de una API REST. Los controladores se ocupan de recibir las peticiones del cliente, invocar a los métodos de los servicios y devolver la respuesta al cliente. Todos hacen uso de la anotación @RestController de Spring que facilita la tarea de crear APIs REST al traducir los objetos Java a objetos JSON en las respuestas enviadas por los métodos.

Para mapear las URL de las peticiones recibidas se emplean las anotaciones @RequestMapping, @PostMapping, @GetMapping, @DeleteMapping y @PutMapping a las que se añade el patrón que debe cumplir la URL. La primera se utiliza a nivel de clase para indicar que las peticiones que cumplan el patrón indicado sean atendidas por la clase anotada, y las otras se emplean a nivel de método para indicar el tipo de la petición atendida, esto es, HTTP POST, GET, DELETE y PUT respectivamente.

En los controladores, al igual que en los servicios, se hace uso de DTOs (*Data Transfer Object*) cuando la información intercambiada entre el cliente y el servidor no puede ser representada mediante ninguna de las entidades definidas en el modelo de datos.

Cliente

La estructura de paquetes del cliente se muestra en la figura 5.13. Cabe destacar que el modelo de variabilidad definido para la LPS condicionó considerablemente la forma en la que fue estructurada la interfaz de usuario y consecuentemente la manera en la que se organizaron los paquetes del cliente web. Para cada paquete se puede destacar:

- **node_modules**: contiene las bibliotecas de npm utilizadas en la aplicación.
- **public**: es la carpeta pública de la aplicación, contiene el archivo index.html.
- **src**: contiene el código de la aplicación. Internamente cuenta con la siguiente jerarquía de paquetes:
  - **assets**: contiene las imágenes de la aplicación, los estilos (CSS) y archivos JavaScript para la internacionalización de mensajes del editor WYSIWYG.
  - **components**: contiene los componentes de la aplicación.
  - **config**: contiene un archivo con información común a varios paquetes.
  - **i18n**: contiene archivos de configuración y de mensajes necesarios para la internacionalización de los textos de la aplicación.
  - **mixins**: contiene archivos con funciones JavaScript de uso común entre componentes de diferentes paquetes.
  - **store**: contiene la configuración de Vuex.
  - **views**: contiene las vistas de la aplicación.

A continuación se detallan los elementos más importantes de los paquetes mencionados.

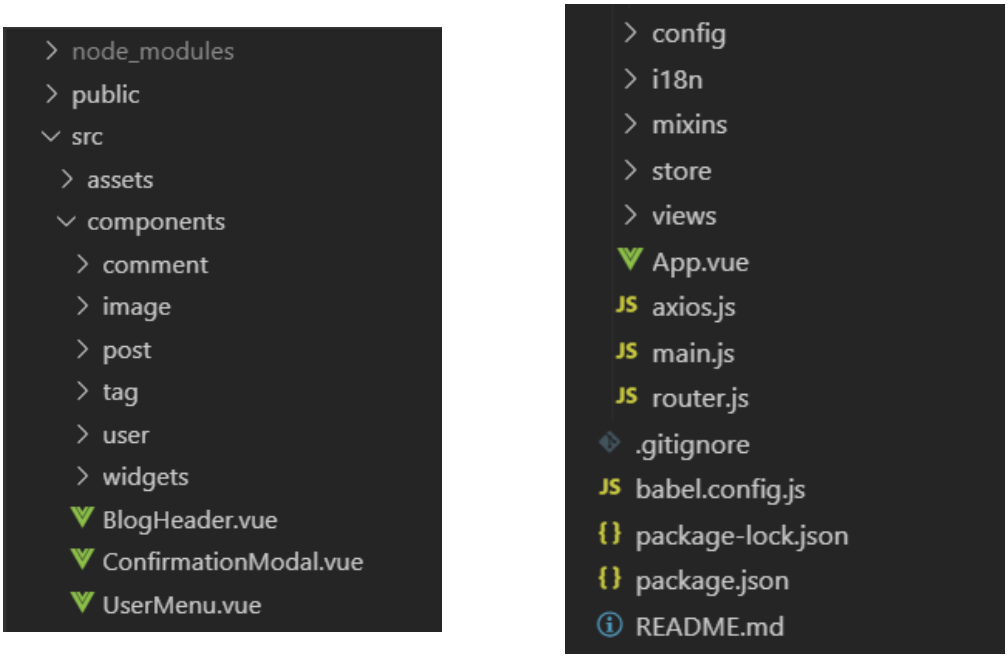


Figura 5.13: Estructura de paquetes del cliente

Componentes. La interfaz web está compuesta de un conjunto de componentes, esto es, instancias de Vue reusables. En la implementación de los componentes se ha optado por emplear *Single File Components*, es decir, cada componente se implementa en un único archivo con extensión .vue formado por tres secciones diferenciadas: *template*, *script* y *style*.

En la primera de ellas, se define la vista del componente empleando una sintaxis basada en HTML que permite vincular el DOM representado a los datos de la instancia de Vue [13]. Este sistema de plantillas facilita la gestión de los diferentes eventos así como la representación de los datos al proporcionar métodos para ejecutar bucles, sentencias condicionales, etc.

En la segunda parte, se define el comportamiento y los datos del componente empleando JavaScript. Aquí se realizan las importaciones de otros componentes o archivos de la aplicación y de las bibliotecas externas. Además, se implementan los métodos que empleará el componente durante su ejecución, por ejemplo, para realizar peticiones al servidor.

En la última sección, se definen los estilos para los elementos que forman parte de la interfaz gráfica del componente, es decir, los elementos de la vista definida en la sección *template*.

Los componentes del cliente se han dividido en los paquetes *views* y *components*, de forma que en el primero de ellos se encuentran aquellos componentes que son la base de una vista y ejercen de contenedores de otros componentes, y en el segundo paquete se encuentran los elementos que forman parte de una vista, o de varias, y son añadidos mediante importaciones a los primeros.

Esta distribución interna de los componentes, así como la organización del paquete *com-*

ponents se ha realizado teniendo en cuenta la variabilidad definida para la línea de producto software, buscando facilitar la labor de anotación realizada posteriormente.

5.2.3 Correspondencias

En esta sección se pretende realizar una revisión del análisis y diseño del producto estableciendo las correspondencias entre las historias de usuario, las peticiones del servicio REST y los métodos de los controladores encargados de atender dichas peticiones. Por motivos de espacio, estas correspondencias se reflejan en el apéndice C.

5.3 Implementación y pruebas

5.3.1 Implementación

En esta sección se tratan los aspectos de implementación más complejos y otras cuestiones relevantes que complementan la información de las secciones anteriores.

Búsqueda de artículos por palabras clave

Una de las funcionalidades definidas para la aplicación es la búsqueda de artículos por palabras clave (historia 42). Para su desarrollo fue necesaria la creación de un nuevo repositorio y la implementación de la búsqueda ya que, pese a sus ventajas, Spring Data no ofrecía ninguna solución para implementar esta consulta. El algoritmo diseñado puede verse en la figura 5.14.

Para su implementación se hizo uso de la interfaz *CriteriaBuilder* de JPA que proporciona un conjunto de funcionalidades para la construcción de consultas. Dentro del algoritmo, primeramente se inicializan el constructor de consultas y la consulta en cuestión, y se determina la clase del elemento sobre el que se realizará la búsqueda (*Post*) y los campos que serán consultados (*title* y *text*).

A continuación, se recorre en un bucle la lista de palabras clave recibidas por parámetro, construyendo predicados para cada una de ellas. En este punto cabe destacar que el hecho de que el contenido de los artículos se almacene en formato HTML en la base de datos duplicó el número de predicados necesarios en la consulta, ya que fue necesario comprobar la existencia de cada palabra en dicho formato y en texto plano. Para realizar la traducción de la palabra recibida a formato HTML se empleó el método *escapeHtml4* de la clase *StringEscapeUtils*.

Seguidamente, se crean dos nuevos predicados, el primero como la unión de los generados en el bucle mediante el operador *or* y el segundo, unido al anterior mediante el operador *and*, con la comprobación de que los artículos tengan fecha de publicación, es decir, no sean borradores.

Posteriormente, se ejecuta la consulta aplicando los predicados definidos anteriormente y se crea una lista con los criterios de ordenación definidos en el parámetro *pageRequest*, para inmediatamente después ordenar los resultados obtenidos.

A continuación, se establecen los parámetros de paginación, esto es, el número de página y el número de resultados por página, en base a los criterios definidos en el parámetro *pageRequest*. Finalmente, se proporciona el resultado mediante una instancia del objeto *Page* que contiene el resultado de la consulta paginado.

LogAspect

Otro de los objetivos definidos para la aplicación es la creación de un *log* que efectúe el registro de los métodos invocados de los controladores, almacenando los valores de entrada y salida de los mismos.

Para su implementación se optó por emplear los aspectos de Spring, esto es, elementos que contienen código transversal a toda la aplicación y que pueden ser inyectados a los métodos mediante el uso de anotaciones.

El aspecto desarrollado (figura 5.15) es una clase Java anotada con @Aspect y @Configuration. Esta clase contiene una variable que representa el *logger* y permitirá la escritura de los mensajes, y dos métodos que implementan el código a ejecutar en los puntos donde se haya inyectado el aspecto. Las anotaciones de dichos métodos (@Before y @AfterReturning) indican el momento en el que deben ser ejecutados, es decir, antes y después del método anotado respectivamente.

Teniendo en cuenta que en el modelo de características definido para la LPS, la funcionalidad de log es opcional, se buscó facilitar el posterior tratamiento de este código indicando en las anotaciones @Before y @AfterReturning las clases en cuyos métodos debía inyectarse el aspecto, en lugar de anotar individualmente cada función de los controladores.

En cuanto a la implementación de los métodos, en ambos se recibe por parámetro un *JoinPoint*, que representa la ejecución del método anotado. A partir de ese parámetro se obtiene la firma del método, la clase a la que pertenece, los argumentos de entrada, en el caso del método *before*, y los argumentos de salida, en el caso del método *afterReturning*. Finalmente, se efectúa la escritura en el *log* de la cadena de texto elaborada en base a esos parámetros.

5.3.2 Pruebas

En esta sección se detallan las pruebas realizadas en la aplicación para verificar el correcto funcionamiento de la misma.

```
public Page<Post> findByKeywords(String[] keywords, Pageable pageRequest) {

    CriteriaBuilder cb = entityManager.getCriteriaBuilder();
    CriteriaQuery<Post> cq = cb.createQuery(Post.class);
    Root<Post> post = cq.from(Post.class);

    Path<String> titlePath = post.get("title");
    Path<String> textPath = post.get("text");

    // Set predicates
    List<Predicate> predicatesKeywords = new ArrayList<>();
    for (String keyword : keywords) {
        String keywordHTML = StringEscapeUtils.escapeHtml4(keyword);
        predicatesKeywords.add(cb.like(cb.lower(titlePath),
                cb.lower(cb.literal("%" + keyword + "%"))));
        predicatesKeywords.add(cb.like(cb.lower(textPath),
                cb.lower(cb.literal("%" + keyword + "%"))));
        predicatesKeywords.add(cb.like(cb.lower(titlePath),
                cb.lower(cb.literal("%" + keywordHTML + "%"))));
        predicatesKeywords.add(cb.like(cb.lower(textPath),
                cb.lower(cb.literal("%" + keywordHTML + "%"))));
    }

    Predicate predicatesOR = cb.or(predicatesKeywords.toArray(
            new Predicate[predicatesKeywords.size()]));
    Predicate predicateAND = cb.and(cb.isNotNull(post.get("publishDate")));

    List<Predicate> predicates = new ArrayList<>();
    predicates.add(predicatesOR);
    predicates.add(predicateAND);

    cq.select(post).where(predicates.toArray(new Predicate[predicates.size()]));
    // Sort result
    List<Order> orderList = new ArrayList<Order>();
    pageRequest.getSort().get().forEach(x -> {
        if (x.getDirection().isAscending()) {
            orderList.add(cb.asc(post.get(x.getProperty())));
        } else if (x.getDirection().isDescending()) {
            orderList.add(cb.desc(post.get(x.getProperty())));
        }
    });
    cq.orderBy(orderList);

    // Create query
    TypedQuery<Post> query = entityManager.createQuery(cq);
    int totalRows = query.getResultList().size();

    query.setFirstResult(pageRequest.getPageNumber() * pageRequest.getPageSize());
    query.setMaxResults(pageRequest.getPageSize());

    Page<Post> result = new PageImpl<Post>(query.getResultList(),
            pageRequest, totalRows);

    return result;
}
```

Figura 5.14: Algoritmo de búsqueda de artículos por palabras clave

```java
@Aspect
@Configuration
public class LogAspect {
    /** The logger. */
    private Logger logger = LoggerFactory.getLogger(this.getClass());

    /**
     * Before.
     *
     * @param joinPoint the join point
     */
    @Before("execution(* es.udc.fic.controller.*.*(..))")
    public void before(JoinPoint joinPoint) {
        MethodSignature signature = (MethodSignature) joinPoint.getSignature();
        String className = signature.getDeclaringType().getName();
        String methodName = signature.getMethod().getName();

        logger.info("Entering method {}() of class {} with parameters: {}",
                methodName, className, Arrays.toString(joinPoint.getArgs()));
    }

    /**
     * After returning.
     *
     * @param joinPoint the join point
     * @param result    the result
     */
    @AfterReturning(value = "execution(* es.udc.fic.controller.*.*(..))",
            returning = "result")
    public void afterReturning(JoinPoint joinPoint, Object result) {
        MethodSignature signature = (MethodSignature) joinPoint.getSignature();
        String className = signature.getDeclaringType().getName();
        String methodName = signature.getMethod().getName();
        logger.info("Method {}() of class {} returned with value: {}",
                methodName, className, result);
    }
}
```

Figura 5.15: Implementación del aspecto para el log

blogen-rest-service

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
es.udc.fic.model.dto		90%		n/a	13	92	26	174	13	92	0	8
es.udc.fic.service		97%		78%	38	159	17	372	0	65	0	7
es.udc.fic.model		96%		100%	8	147	13	296	8	112	0	7
es.udc.fic.controller		97%		81%	3	52	10	270	0	44	0	5
es.udc.fic.repository		96%		66%	2	6	1	33	0	3	0	1
es.udc.fic.blogen		37%		n/a	1	2	2	3	1	2	0	1
es.udc.fic.utils		98%		100%	1	18	1	47	1	16	1	6
es.udc.fic.security		100%		75%	1	15	0	92	0	13	0	4
Total	166 of 5.682	97%	44 of 278	84%	67	491	70	1.287	23	347	1	39

Figura 5.16: Informe generado por JaCoCo

Pruebas automáticas

Las pruebas automáticas verifican la correcta implementación de las funcionalidades desarrolladas mediante la ejecución de los métodos y la comparación del resultado obtenido con el esperado para un conjunto de valores de entrada.

Para el desarrollo de estas pruebas se han empleado las utilidades y anotaciones proporcionadas por Spring mediante el uso del módulo *spring-boot-starter-test*. Este módulo proporciona una serie de bibliotecas para la realización de pruebas, de las cuales se han empleado: JUnit 4, Spring Test y Spring Boot Test.

Para la ejecución de las pruebas se ha utilizado JUnit, un *framework* Java de código abierto para la implementación y ejecución de pruebas; y se ha complementado con el uso de JaCoCo, una biblioteca de código abierto para Java que permite analizar la cobertura de código de las pruebas [26]. Ambos se han integrado con Maven de forma que mediante la instrucción *mvn test jacoco:report* se ejecutan todas las pruebas y se genera un informe en el directorio /target/site/jacoco/index.html con el resultado del análisis realizado por JaCoCo, que puede verse en la figura 5.16.

En este informe se distingue entre la cobertura de instrucciones y la cobertura de ramas. La primera indica las líneas de código que se han ejecutado al menos una vez durante la ejecución de las pruebas, mientras que la segunda indica el porcentaje de ramas que se han ejecutado para cada sentencia de decisión.

Este tipo de pruebas se han realizado sobre los servicios ofrecidos por la API REST mediante el uso de MockMvc, un *framework* para la realización de pruebas sobre los controladores. Este *framework* utiliza *mocks* para simular el comportamiento de un contenedor de aplicaciones y proporciona soporte para cargar la configuración de Spring en las pruebas mediante el *framework* TestContext [41].

Debido a la estructura interna del servidor en capas, probar los servicios ofrecidos por los controladores implica probar también la implementación de las funcionalidades de los servicios y de los métodos ofrecidos por los repositorios, de forma que se reduce el número de pruebas a realizar pero no se disminuye la cobertura de las mismas.

```
@Test
@Sql("classpath:create-data.sql")
public void testDeletePostAdmin() throws Exception {
    // Prepare post
    String token = loginAdminUser();
    String title = "post title";
    String text = "<h3>Post text content</h3>";
    Post post = new Post(title, text, null);

    // Save post
    MvcResult pResult = mvc
            .perform(post("/post/save").content(asJsonString(post)).contentType(MediaType.APPLICATION_JSON)
                    .header(HttpHeaders.AUTHORIZATION, token).accept(MediaType.APPLICATION_JSON))
            .andExpect(status().isCreated()).andReturn();
    Post postResult = new ObjectMapper().readValue(pResult.getResponse().getContentAsString(), Post.class);

    // Find post before delete
    mvc.perform(get("/post/find").param("postId", postResult.getPostId().toString())
            .contentType(MediaType.APPLICATION_JSON).accept(MediaType.APPLICATION_JSON)).andExpect(status().isOk())
            .andExpect(MockMvcResultMatchers.jsonPath("$.postId").value(postResult.getPostId()))
            .andExpect(MockMvcResultMatchers.jsonPath("$.title").value(title))
            .andExpect(MockMvcResultMatchers.jsonPath("$.text").value(text))
            .andExpect(MockMvcResultMatchers.jsonPath("$.updateDate").isNotEmpty());

    // Delete post
    mvc.perform(delete("/post/delete").param("postId", postResult.getPostId().toString())
            .contentType(MediaType.APPLICATION_JSON).header(HttpHeaders.AUTHORIZATION, token)
            .accept(MediaType.APPLICATION_JSON)).andExpect(status().isOk());

    // Find post after delete
    mvc.perform(get("/post/find").param("postId", postResult.getPostId().toString())
            .contentType(MediaType.APPLICATION_JSON).accept(MediaType.APPLICATION_JSON))
            .andExpect(status().isNotFound());
}
```

Figura 5.17: Ejemplo de prueba automática

En la figura 5.17 puede verse un ejemplo de una de las pruebas realizadas. MockMvc permite establecer los parámetros de la petición realizada sobre el servicio REST y verificar el contenido de la respuesta obtenida. En el ejemplo se muestran diferentes tipos de peticiones. La primera de ellas es una petición POST, cuya finalidad es guardar un artículo; la segunda es una petición GET para buscar el artículo; y la tercera es una petición DELETE, ejecutada para eliminar el artículo.

Para todas ellas se especifica el tipo de la petición (GET, POST, PUT, DELETE) a ejecutar y el tipo del contenido. Además, se incluye la cabecera AUTHORIZATION, en la que se envía el token del usuario autenticado, en las peticiones POST y DELETE; el parámetro *postId* en las peticiones DELETE y GET; y el artículo a crear en formato JSON, en el contenido de la petición POST.

Una vez definidos los parámetros de la petición enviada, se establecen los valores esperados para la respuesta recibida, especificando el código HTTP y el contenido de los objetos incluidos en la respuesta.

Además de las pruebas realizadas sobre los controladores, también se han realizado pruebas sobre las entidades para comprobar la correcta implementación de los métodos *equals* y *hashcode*, verificando que la comparación entre entidades se hiciese por contenido y no por referencia.

En total se han desarrollado 216 test en los que se han probado las funcionalidades imple-

```
imagesFolder=C:\\Users\\Public\\Pictures\\Test\\
idToken=
idToken2=
invalidIdToken=
invalidIdTokenEmail=
```

Figura 5.18: Parámetros necesarios para la ejecución de las pruebas

mentadas en la aplicación, verificando tanto su correcta ejecución como los casos de error.

Configuración de las pruebas Parte de las pruebas automáticas implementadas requieren la configuración de una serie de parámetros para su correcta ejecución. Dichos parámetros se encuentran en el fichero /src/test/resources/config.properties (figura 5.18) y las instrucciones para su configuración se han incorporado al archivo README.md del proyecto en el que se ha implementado el servidor de la aplicación.

Por una parte, se encuentran los tests que prueban las funcionalidades relacionadas con la gestión de imágenes. Estas pruebas requieren que se especifique una carpeta en la que se guardarán las imágenes generadas durante la ejecución de los tests (parámetro *imagesFolder*).

Por otra parte, se encuentran las pruebas sobre las funcionalidades de creación, modificación y borrado de los comentarios realizados por usuarios de Google. La implementación de estas funcionalidades se ha hecho empleando tokens de autenticación de Google pero, debido a que su periodo de expiración es de una hora, no ha sido posible especificar valores estáticos para ellos en las pruebas y es necesario configurarlos mediante el uso de parámetros.

Como puede verse en la figura 5.18, se requieren cuatro tipos de tokens, cuya generación se realiza mediante una herramienta para desarrolladores ofrecida por Google y a la que se puede acceder a través de la siguiente url: https://developers.google.com/oauthplayground/.

En el archivo README.md se ha detallado el procedimiento a seguir a la hora de generar los tokens, incluyendo capturas de pantalla con el fin de facilitar el proceso.

Pruebas manuales

Además de las pruebas automáticas, se han llevado a cabo multitud de pruebas manuales sobre la aplicación.

En la realización de este tipo de pruebas se ejercitaron las posibles interacciones de un usuario con la interfaz web y se comprobó que el cliente se comportase correctamente. Se verificó que ejecutase las peticiones apropiadas al servidor, que mostrase la información de forma correcta, que se cumpliesen las restricciones de acceso a las páginas en base al rol del usuario autenticado, que la navegación entre estas fuese adecuada, etc.

Capítulo 6

Construcción de la línea de producto software

6.1 Análisis

En esta sección se describen aquellos aspectos del análisis de la línea de producto software no tratados en el capítulo 4.

6.1.1 Arquitectura del sistema

En esta sección se describe la arquitectura general de la línea de producto software, realizando una descomposición de alto nivel de los componentes que la conforman. En la figura 6.1 puede verse un esquema de la arquitectura diseñada, cuyos principales elementos son los siguientes:

- **Interfaz web**: es el componente que muestra las funcionalidades de la herramienta al usuario. A través de la interfaz se pueden seleccionar las características para el producto, importar o exportar un fichero de especificación, añadir un título para el producto y generar un blog.

- **Componentes**: son los elementos reutilizables que implementan las distintas funcionalidades y características definidas para el producto.

- **Motor de derivación**: es el elemento encargado de generar el código del producto a partir de los componentes y en base a la especificación definida para el producto, es decir, las características seleccionadas.

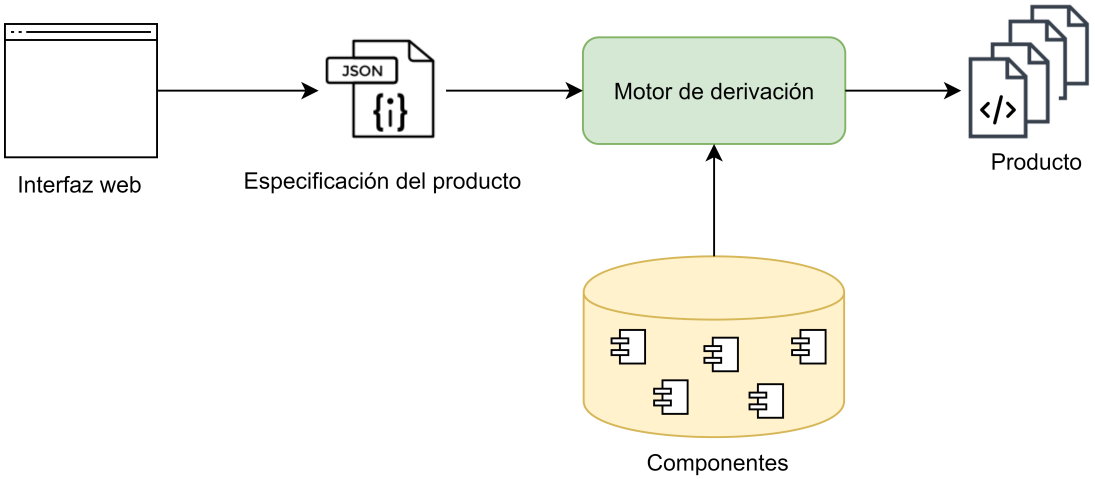


Figura 6.1: Arquitectura general de la línea de producto Software

6.1.2 Interfaz de usuario

En esta sección se pretende realizar una descripción de la interfaz de usuario de la herramienta de generación.

La interfaz web diseñada es muy simple y está formada por una única pantalla, que puede verse en la figura 6.2. Dentro del contenido de la página se muestran diferentes elementos.

En la parte superior se encuentra la cabecera con el nombre de la herramienta de generación y, a continuación, un campo de texto que permite introducir el título del blog que se generará.

En la parte izquierda, se muestra un árbol con casillas de selección en el que se exponen todas las características disponibles para el producto y a través del cual se pueden seleccionar o quitar las funcionalidades deseadas para el producto.

En la parte derecha se muestran dos botones, el primero de ellos permite importar un fichero de especificación de un producto, y el segundo ofrece la opción de exportar en un archivo JSON la especificación del producto definida.

Por último, en la parte inferior se muestra el botón que permite generar un producto en base a las características y al título determinados.

6.2 Diseño

6.2.1 Arquitectura tecnológica del sistema

En esta sección se pretende complementar el análisis de la arquitectura realizado en la sección 6.1.1, a través del estudio de las tecnologías empleadas en cada componente del sistema. La figura 6.3 ilustra la arquitectura tecnológica de la LPS.

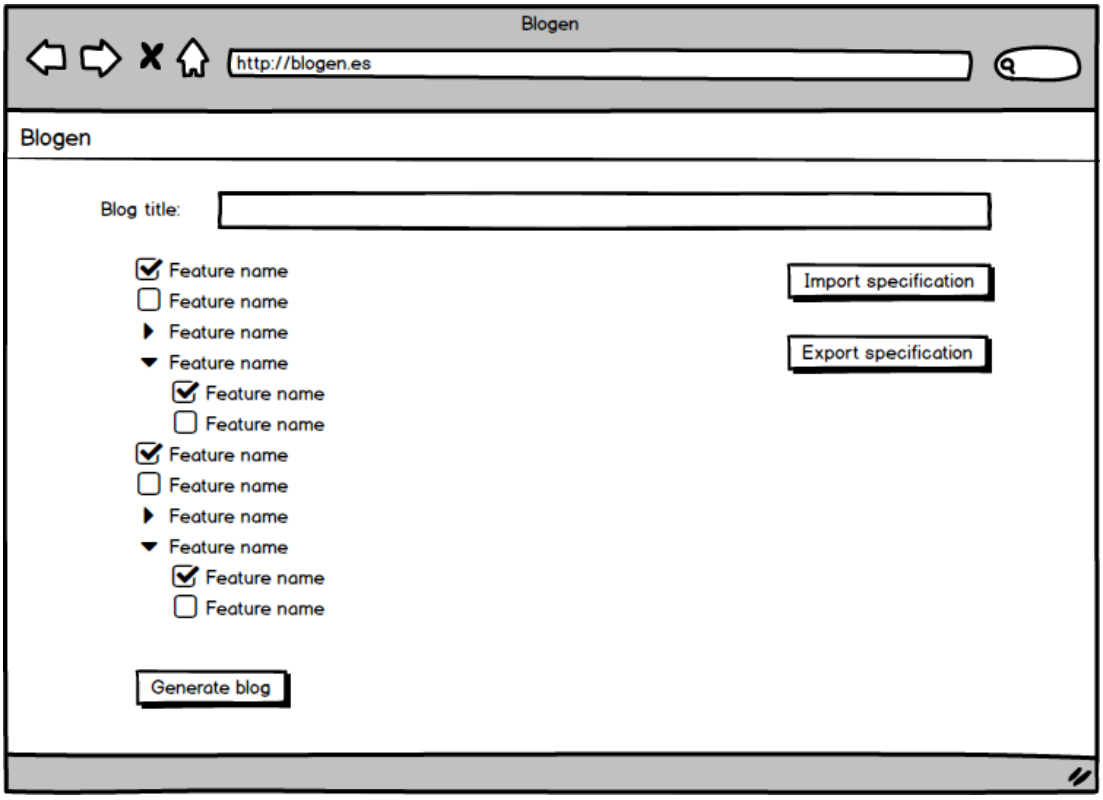


Figura 6.2: Pantalla principal de la herramienta de generación

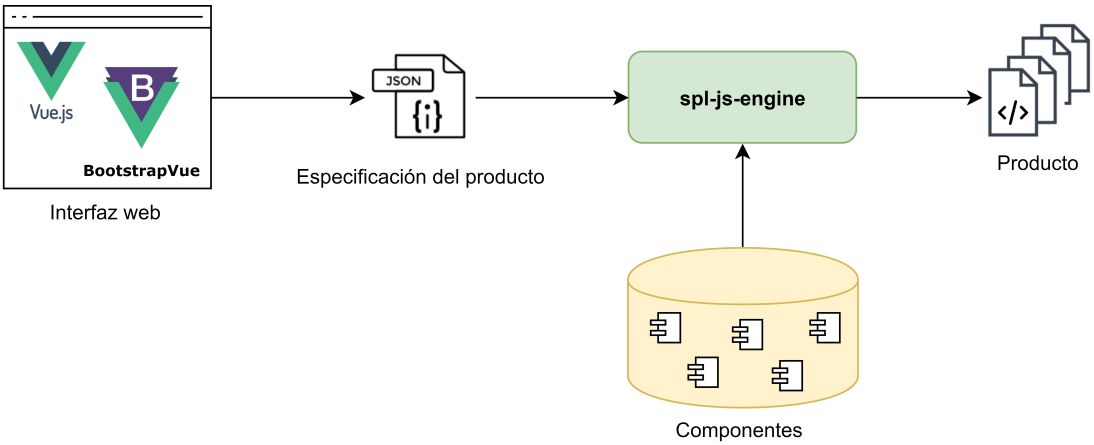


Figura 6.3: Arquitectura tecnológica de la línea de producto software

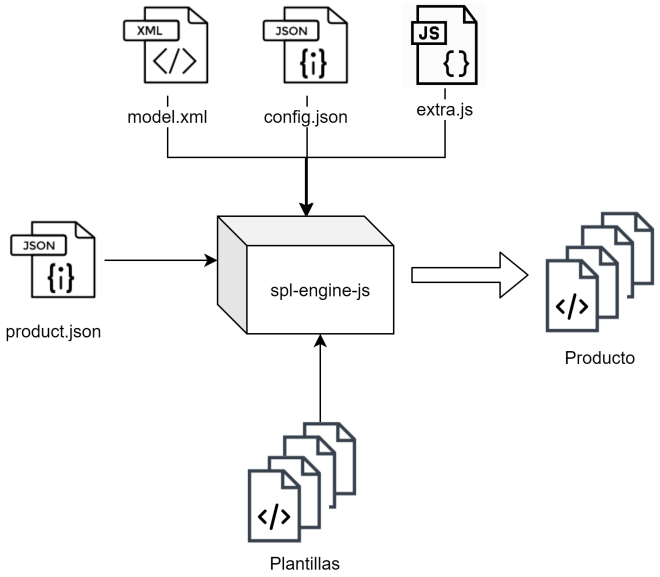


Figura 6.4: Esquema del motor de derivación spl-js-engine

Motor de derivación

Para la generación del código fuente de los productos de la LPS se ha empleado spl-js-engine, un motor de derivación JavaScript basado en *scaffolding*. Esta herramienta hace uso de anotaciones sobre el código original del producto (plantillas) para definir los fragmentos que pertenecen a cada componente, en lugar de separar físicamente el código de cada uno. Esto supone una ventaja sobre los enfoques compositivos ya que simplifica el proceso de desarrollo de la LPS. Las anotaciones en las plantillas se indican mediante comentarios del lenguaje de programación de la plantilla y su contenido es cualquier código JavaScript. Además, el motor de derivación permite usar variables en las plantillas, que serán sustituidas en el producto por los valores indicados [10].

En la figura 6.4 puede verse un esquema de los elementos de entrada que recibe el motor de derivación para generar un producto, y que se describen a continuación.

- **model.xml**: fichero que contiene el modelo de características de la línea de producto software. Para la creación y modificación de este archivo se empleó FeatureIDE, un plugin de Eclipse que permite definir modelos de características en formato XML y generar su representación gráfica (figura 4.2). En el apéndice D puede verse el archivo XML que representa el modelo de características definido para la LPS de este proyecto.
- **config.json**: fichero que contiene la configuración del motor de derivación. En él se definen los delimitadores de las anotaciones en función de la extensión de los archivos, y los archivos y carpetas que se deben ignorar a la hora de generar los productos.

```
{
    "features": [
        "Blog",
        "Post",
        "UserManagement",
        "PostEditor",
        "EditorType",
        "Images",
        "Tags",
        "ImageUploading",
        "ImageFromURL",
        "AnonymousRegistration",
        "Internationalization",
        "Widgets",
        "TagsWidget",
        "FileWidget",
        "CommentsWidget",
        "Comments",
        "SearchWidget",
        "GoogleComments",
        "RegisteredComments",
        "AnonymousComments",
        "WYSIWYGEditor",
        "Logger"
    ],
    "data": {
        "title": "Blog"
    }
}
```

Figura 6.5: Ejemplo de fichero con la especificación de un producto

- **extra.js**: fichero que contiene código JavaScript que se puede emplear al derivar un producto.

- **product.json**: fichero que contiene la especificación de un producto en formato JSON. En la figura 6.5 puede verse el contenido del fichero de especificación de un producto.

- **Plantillas**: código fuente anotado, es decir, el código fuente desarrollado para el producto (capítulo 5) con las anotaciones correspondientes.

Interfaz web

Para el desarrollo de la interfaz web se han empleado Vue.js y BootstrapVue, y además, se ha hecho uso de la biblioteca vue-router. Estas tecnologías también fueron utilizadas en el desarrollo del cliente web del producto, por lo que su descripción puede verse en la sección 5.2.1.

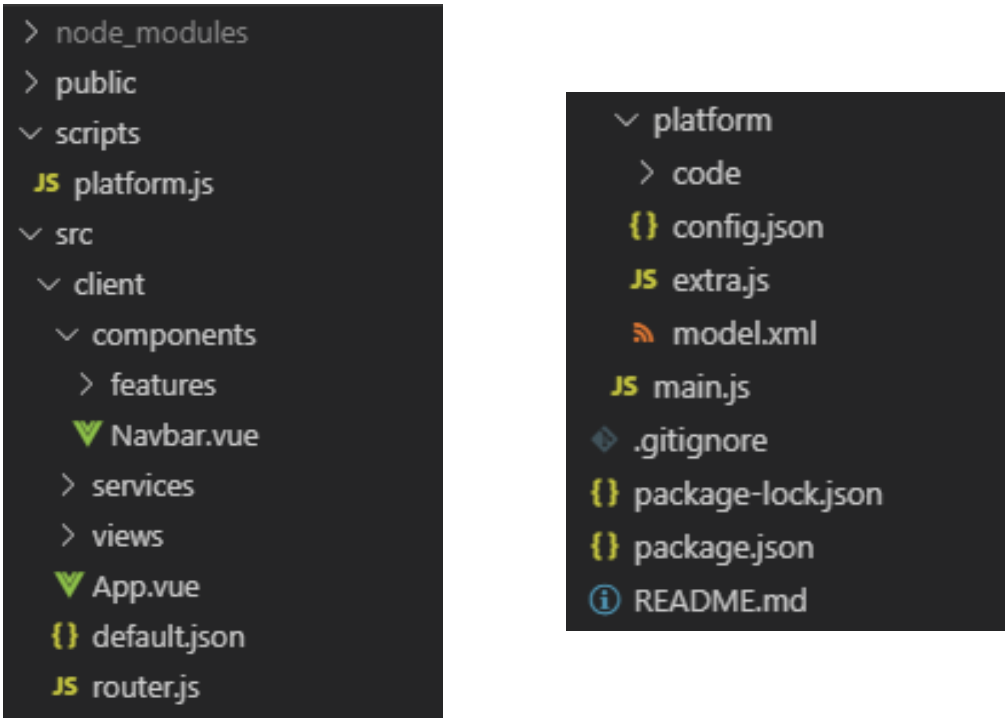


Figura 6.6: Estructura de paquetes de la herramienta de generación

6.2.2 Diseño de la aplicación

En esta sección se detallará la estructura interna de la aplicación web desarrollada para la herramienta de generación.

En la figura 6.6 puede verse la estructura de paquetes establecida para la herramienta. Para cada uno de ellos se puede destacar:

- **node_modules**: contiene las bibliotecas de npm utilizadas en la aplicación.
- **public**: es la carpeta pública de la aplicación, contiene el archivo index.html.
- **scripts**: contiene archivos JavaScript auxiliares.
- **src**: contiene el código de la aplicación y los elementos necesarios para generar productos. Internamente cuenta con la siguiente jerarquía de paquetes:
  - **client**: contiene el código de la aplicación web. Se divide en:
    - **components**: contiene los componentes que forman parte de la aplicación.
    - **services**: contiene archivos JavaScript que implementan las funcionalidades y el comportamiento de la aplicación haciendo uso del motor de derivación.
    - **views**: contiene las vistas de la aplicación.

- **platform**: contiene los archivos necesarios para la generación de productos, es decir, aquellos que recibe como entradas el motor de derivación. La carpeta *code* alberga las plantillas.

La estructura de los componentes de esta aplicación es idéntica a la definida en la sección 5.2.2 ya que, al igual que para el desarrollo del cliente web del producto, se han empleado *Single File Components*.

Cabe destacar que la interfaz web de la herramienta de generación fue desarrollada a partir de un código de ejemplo proporcionado por los directores del trabajo, por lo que parte de los archivos de esta estructura no han sido implementados por la autora del proyecto.

6.3 Implementación y pruebas

6.3.1 Implementación

En esta sección se tratan los aspectos de implementación más relevantes.

Anotaciones

El aspecto más relevante en la implementación de la línea de producto software fue la anotación del código desarrollado para el producto.

Como se ha explicado en la sección 6.2.1, las anotaciones empleadas por el motor de derivación se incluyen en el código fuente del producto como comentarios del lenguaje de programación de cada plantilla y su contenido es código JavaScript.

A la hora de anotar el código fuente, fue necesario definir en el archivo de configuración (config.json) los delimitadores de los comentarios para cada tipo de plantilla. Dicha configuración puede verse en la figura 6.7.

Este tipo de configuración permite establecer para cada extensión de archivo un delimitador diferente, posibilitando de esta forma que las anotaciones se integren como comentarios del lenguaje empleado y no afecten a la hora de compilar el código. Sin embargo, en los archivos con extensión .vue, no es posible definir un delimitador de comentario que sea válido en todas las secciones del fichero (*template*, *script* y *style*), ya que en cada una de ellas se emplea un lenguaje diferente (HTML, JavaScript y CSS) y el formato de los comentarios es distinto para cada uno de ellos. Además, también se generan errores de compilación en ficheros que no admiten comentarios, como los archivos JSON.

Una vez establecida la configuración, se procedió a anotar el código fuente del servidor y del cliente web del producto desarrollado. Para ello, se tuvo en cuenta que el motor de derivación empleado se encarga de realizar las comprobaciones necesarias de las dependencias

```
"delimiters": [
    {
        "extension": [
            "html",
            "jsp",
            "xml"
        ],
        "start": "<!--%",
        "end": "%-->"
    },
    {
        "extension": [
            "js",
            "java",
            "vue",
            "json",
            "sql",
            "md",
            "css"
        ],
        "start": "/*%",
        "end": "%*/"
    }
]
```

Figura 6.7: Configuración para los delimitadores de las anotaciones

y restricciones establecidas en el modelo de características, por lo que no fue necesario realizar dichas comprobaciones en las anotaciones.

En la figura 6.8 se muestra un ejemplo de las anotaciones realizadas sobre el código del cliente web del producto. Como puede verse, se determina un fragmento de código que solo debe ser añadido al producto generado si se ha seleccionado al menos una de las características definidas (*FileWidget*, *TagsWidget* o *CommentsWidget*). Dentro de dicho fragmento se establecen nuevas restricciones para definir el código que está asociado a cada característica.

Por otra parte, en la figura 6.9 se muestra un ejemplo de las anotaciones realizadas sobre el código del servidor web del producto. Como puede verse, las anotaciones se emplean para delimitar los fragmentos de código dentro del método *deletePost* que dependen de la existencia de imágenes (*Images*) y de la existencia de etiquetas (*Tags*).

Por último, recordar que en la sección 6.2.1 se mencionó que el motor de derivación permite usar variables en las plantillas. El valor de las variables se define en el fichero de especificación del producto y mediante anotaciones se especifica el lugar donde deben substituirse su valor. Un ejemplo de este tipo de anotaciones puede verse a continuación.

```
1 "blog": /*%= data.title %*/
```

```
/*% if (feature.FileWidget || feature.TagsWidget || feature.CommentsWidget) { %*/
<b-col :lg="widgetsCol" md="12" sm="12">
  /*% if (feature.SearchWidget) { %*/
  <SearchWidget class="blog-widget"></SearchWidget>
  /*% } %*/
  /*% if (feature.FileWidget) { %*/
  <ArchiveWidget class="blog-widget"></ArchiveWidget>
  /*% } %*/
  /*% if (feature.TagsWidget) { %*/
  <TagsWidget class="blog-widget"></TagsWidget>
  /*% } %*/
  /*% if (feature.CommentsWidget) { %*/
  <CommentsWidget class="blog-widget"></CommentsWidget>
  /*% } %*/
</b-col>
/*% } %*/
```

Figura 6.8: Ejemplo de anotaciones en el cliente del producto

```
public void deletePost(long postId, UserProfile user) throws InstanceNotFoundException,
        IllegalAccessException {
    Post post = findById(postId);
    if ((user.getRole() == Role.ADMIN) || (post.getUser() == user)) {
        /*% if (feature.Images) { %*/
        // Delete images
        for (Image image : post.getImages()) {
            if (image.getPosts().size() == 1 && image.getPosts().contains(post)) {
                imageService.deleteImage(image);
            }
        }
        /*% } %*/

        /*% if (feature.Tags) { %*/
        // Delete tags
        for (Tag tag : post.getTags()) {
            if (tag.getPosts().size() == 1 && tag.getPosts().contains(post)) {
                tagService.deleteTag(tag);
            }
        }
        /*% } %*/
        postRepository.delete(post);
    } else {
        throw new IllegalAccessException("Unable to delete post");
    }
}
```

Figura 6.9: Ejemplo de anotaciones en el servidor del producto

6.3.2 Pruebas

A la hora de probar el correcto desarrollo de la línea de producto software, se realizaron numerosas pruebas manuales.

Con el fin de verificar que las anotaciones definidas para las plantillas eran correctas, se generaron múltiples productos, pero, debido al gran número de elementos del modelo de características diseñado, no fue factible probar todas las combinaciones posibles de características. Sin embargo, se intentó probar aquellas más relevantes de forma que se verificase la corrección de todas las anotaciones incluidas en el código.

Además, sobre la interfaz web desarrollada, se realizaron pruebas para verificar que se generaban, comprimían y descargaban correctamente los archivos zip con el código fuente del producto generado; que era posible exportar un fichero de especificación en base a las características seleccionadas; que se actualizaba correctamente la vista del árbol de características al importar un fichero de especificación; que el título introducido se establecía correctamente en el blog generado, etc.

Capítulo 7

Solución desarrollada

En este capítulo se pretenden mostrar las características más relevantes de la línea de producto software desarrollada. Inicialmente se tratarán los aspectos relacionados con la herramienta de generación, y posteriormente se mencionarán las características del producto.

7.1 Herramienta de generación

En esta sección se ilustrará el funcionamiento de la herramienta de generación.

En la figura 7.1 puede verse la página principal de la herramienta. En ella se muestran todas las características disponibles para el producto en forma de árbol junto a las casillas de selección que permiten añadir o eliminar cada característica del producto. A la hora de seleccionar las características se comprueban las restricciones y dependencias definidas en el modelo de características y se muestran los errores correspondientes. En la figura 7.2 puede verse un ejemplo de un error en la selección.

Además, en la parte derecha se incluyen las funcionalidades de importación y exportación del fichero de especificación de un producto y el campo de texto para la introducción del título del blog.

Por último, debajo de la estructura de características se muestra el botón que permite generar un producto. Tanto este botón, como el de exportación de un fichero de especificación, solo se muestran habilitados cuando no existen errores en la selección.

Al pulsar en el botón de generar un blog, se genera un archivo zip con el código fuente del producto y se descarga automáticamente.

7.2 Producto

En esta sección se ilustrarán los aspectos principales del producto de la LPS, mostrando capturas de pantalla de un producto con todas las características posibles.

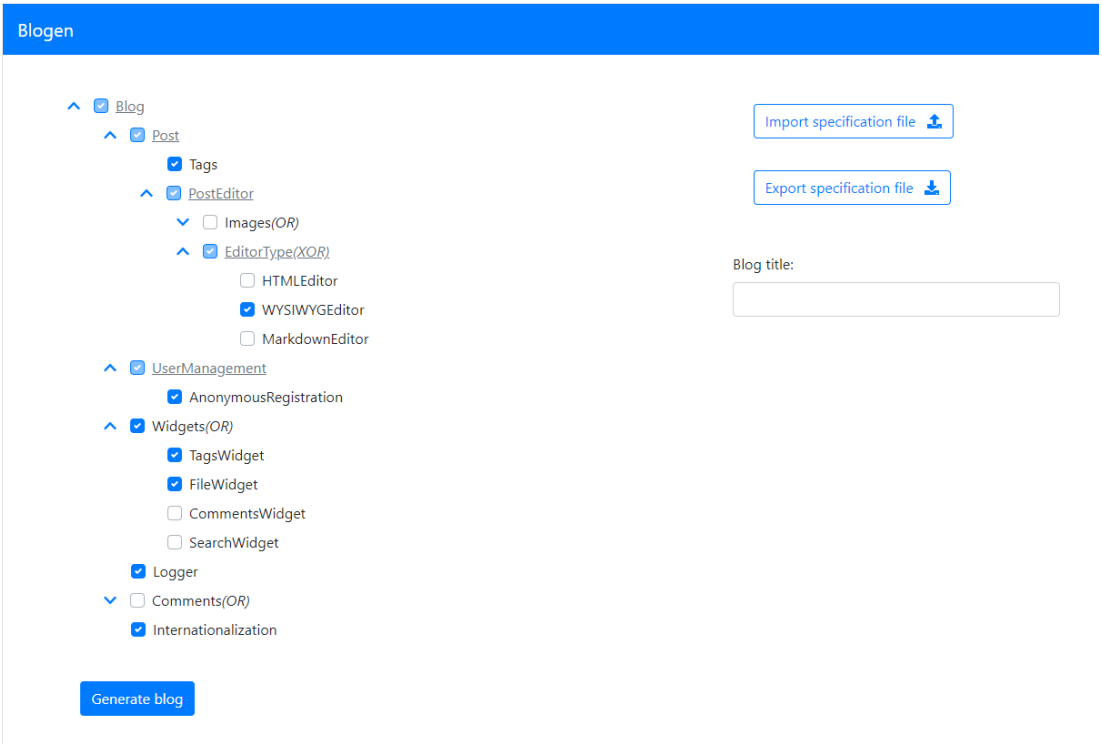


Figura 7.1: Página principal de la herramienta de generación

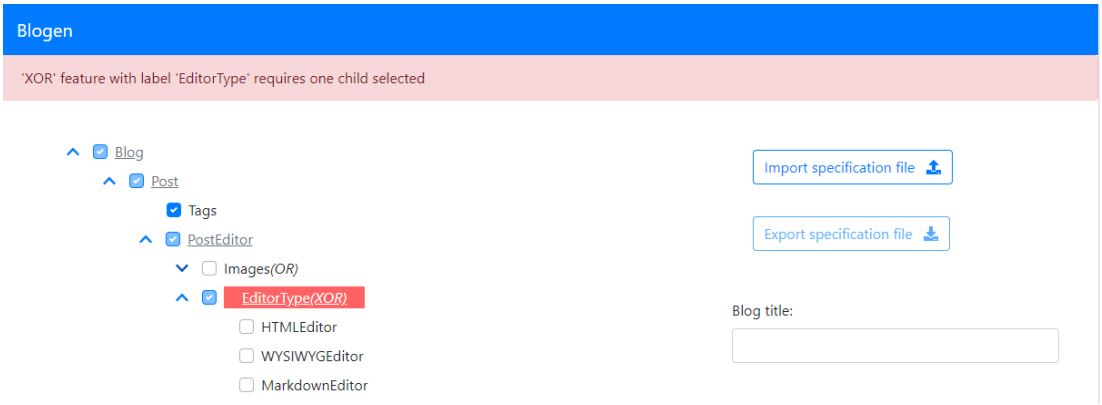


Figura 7.2: Página principal de la herramienta con error en la selección

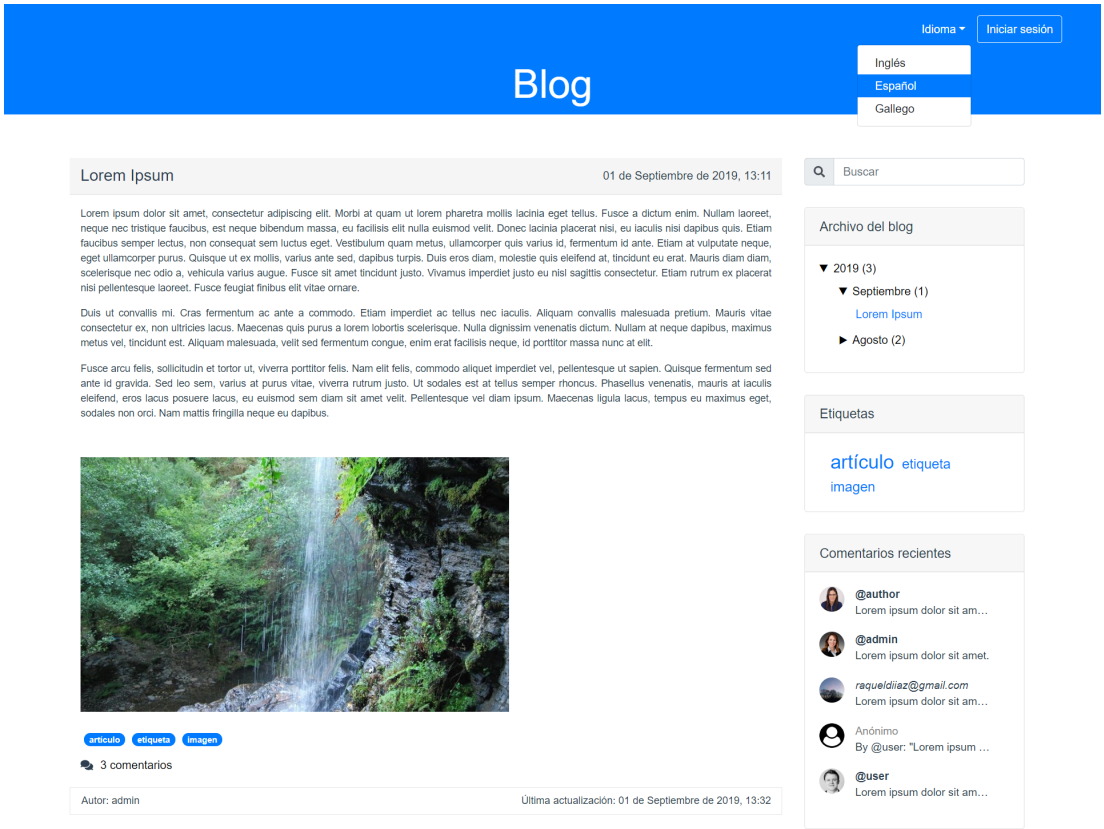


Figura 7.3: Página principal del blog

La página principal del blog se muestra en la figura 7.3. En la parte superior puede verse la cabecera de las páginas públicas con el selector de idioma y el botón para acceder a la página de inicio de sesión (figura 7.4) desde la que, a su vez, se puede acceder a la página de registro (figura 7.5).

El contenido se divide en dos columnas. La primera de ellas muestra los artículos publicados con su información asociada, en este caso se muestra un artículo con texto, imágenes, comentarios y etiquetas. En la segunda columna se muestran los *widgets* del blog, es decir, la barra de búsqueda, el archivo del blog, la nube de etiquetas y los comentarios recientes.

Para acceder a la búsqueda de artículos por palabras clave, basta con introducir los términos por los que se desea buscar en la barra de búsqueda y pulsar la tecla *enter*. Los resultados obtenidos se muestran como se puede ver en la figura 7.6.

De forma prácticamente idéntica se muestran los resultados de la búsqueda de artículos por etiqueta, como se puede ver en la figura 7.7. A esta búsqueda se puede acceder pulsando en el nombre de una etiqueta asociada a un artículo o mediante la nube de etiquetas.

Pulsando en el título del artículo se accede a la página que muestra el contenido completo

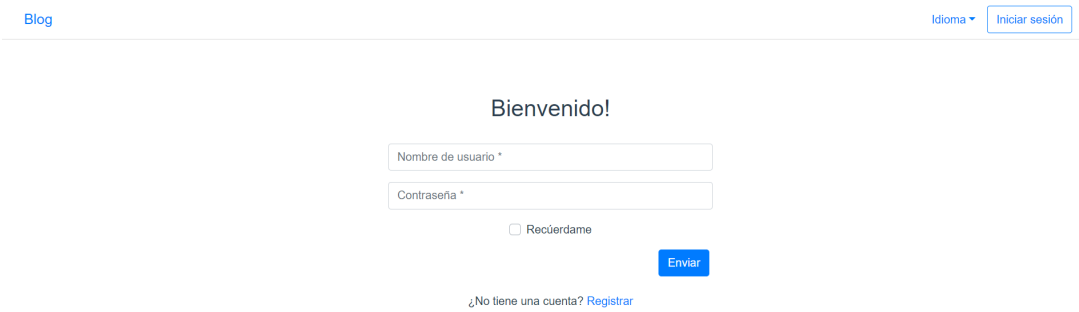


Figura 7.4: Página de inicio de sesión

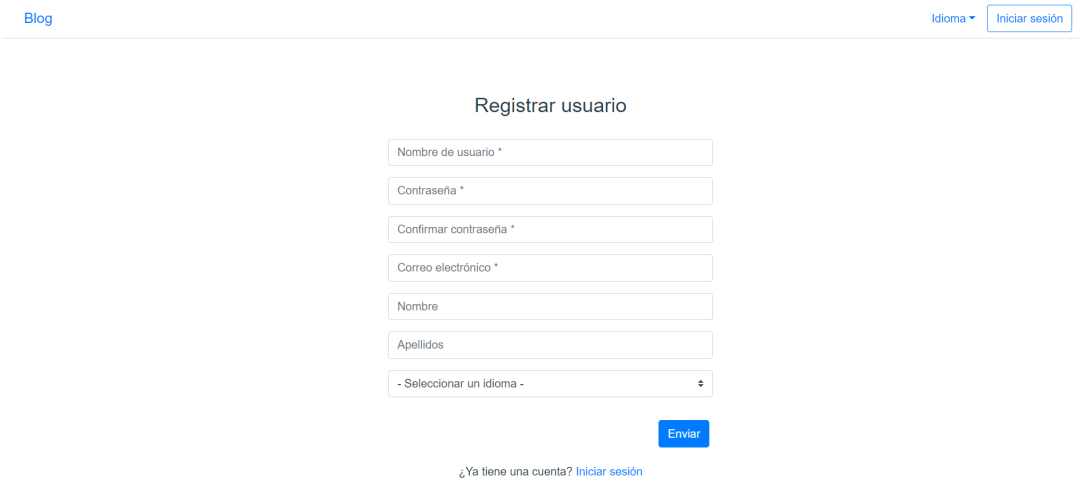


Figura 7.5: Página de registro

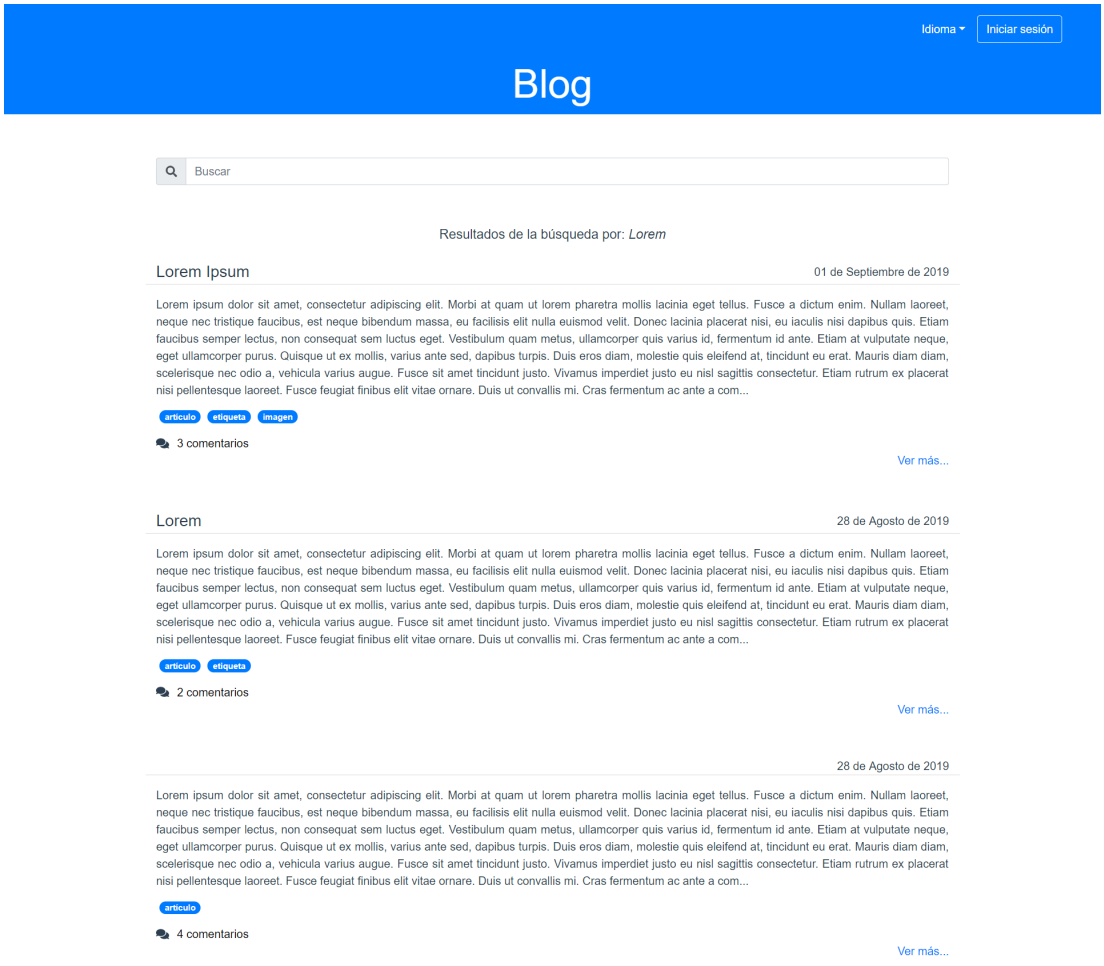


Figura 7.6: Página con los resultados de la búsqueda por palabras clave

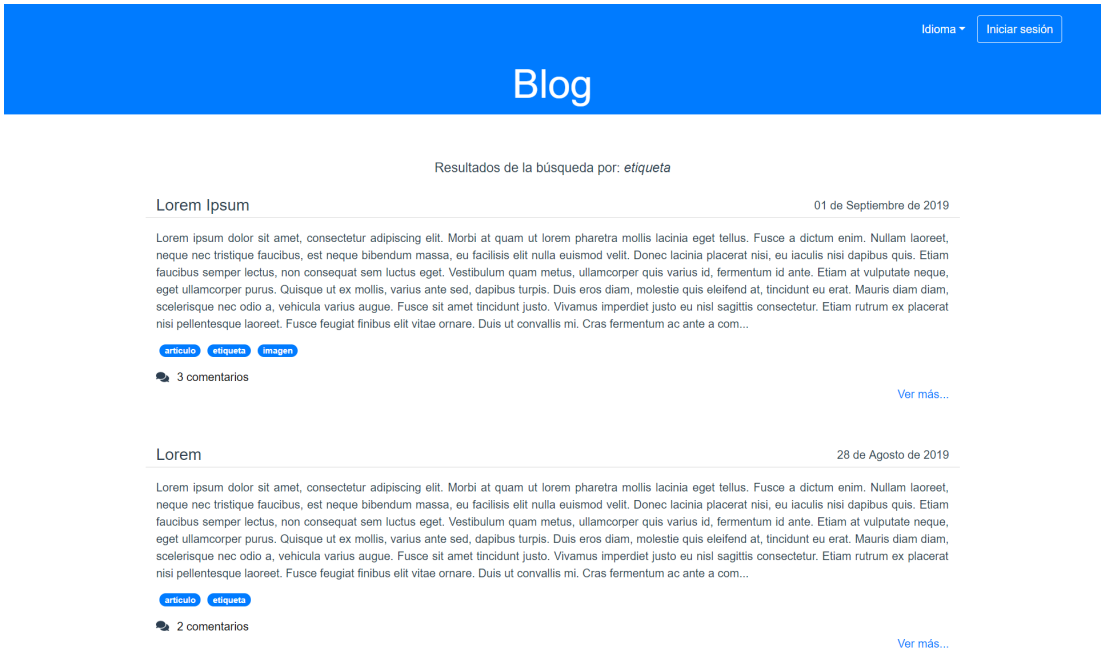


Figura 7.7: Página con los resultados de la búsqueda por etiqueta

del artículo en cuestión, y que puede verse en la figura 7.8. En dicha figura se muestra un ejemplo de un artículo con imágenes, etiquetas y comentarios, cada uno de los cuales es de un tipo diferente. El comentario más antiguo ha sido realizado por un usuario registrado en la aplicación, el siguiente por un usuario anónimo, y el más reciente por un usuario de Google.

Desde esta vista, es posible acceder al perfil del usuario autor del artículo pulsando en su imagen o nombre de usuario. A la hora de mostrar el perfil de un usuario se distinguen dos casos, si el perfil pertenece a un usuario distinto del autenticado o no hay ningún usuario autenticado se muestra el perfil público del autor (figura 7.9), mientras que si el perfil pertenece al usuario autenticado se muestra su perfil privado (figura 7.10).

Un usuario autenticado puede acceder en cualquier momento a su perfil privado a través de la opción "Perfil" del menú de usuario (mostrado en la cabecera de la figura 7.10). Desde la página de su perfil privado, un usuario puede acceder a la modificación de este, a la modificación de la contraseña asociada a su cuenta y a la desactivación de su cuenta empleando los botones correspondientes.

El menú de usuario también proporciona acceso a la página de gestión del blog mediante la opción "Gestión del blog". Al acceder a esta vista, se muestra por defecto la pestaña de gestión de artículos (figura 7.11). En ella pueden verse los artículos publicados y guardados (borradores) del blog. Para cada artículo se muestra el título, una marca que indica si se trata de

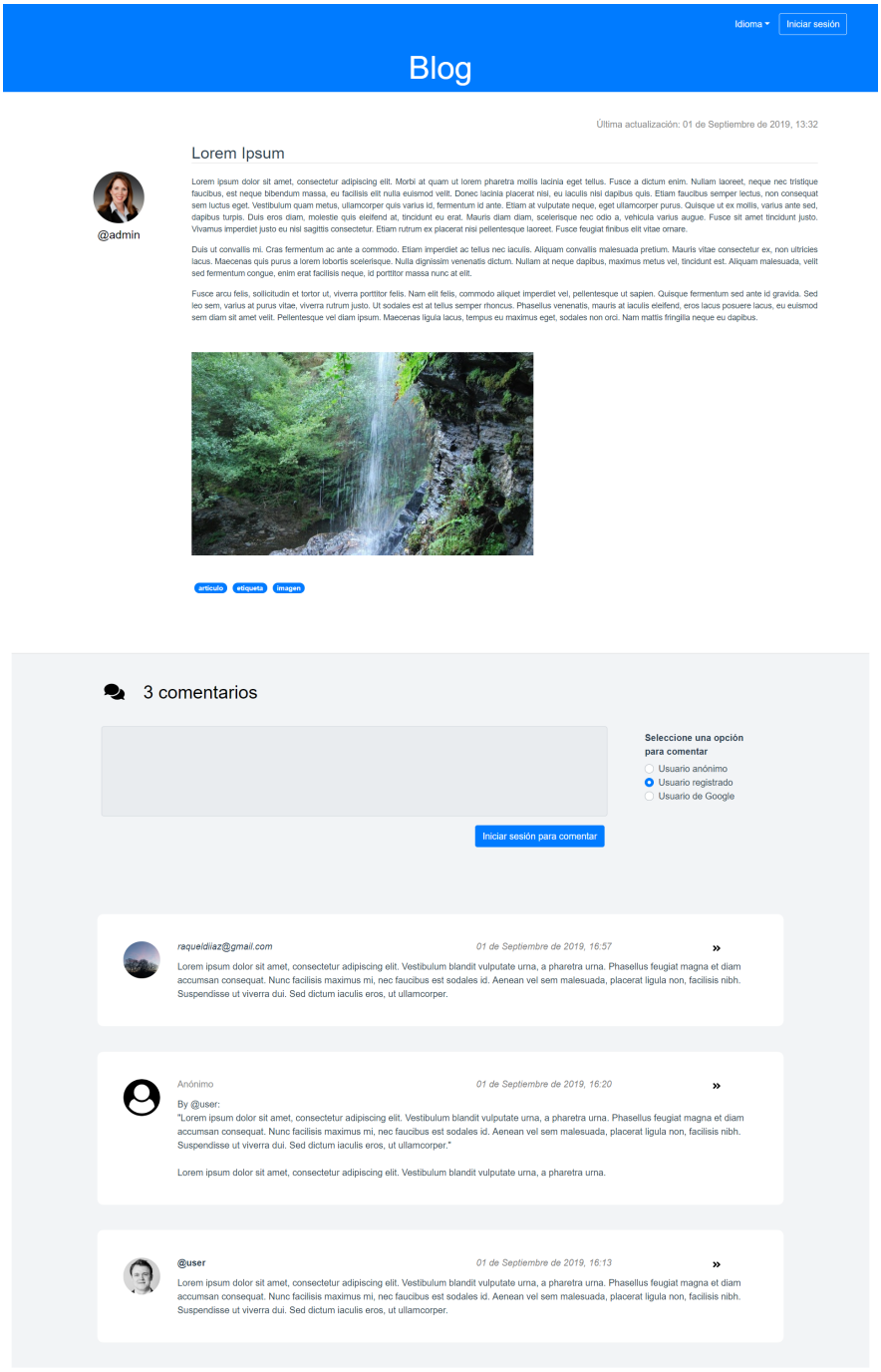


Figura 7.8: Página de un artículo

Figura 7.9: Página del perfil público de un usuario

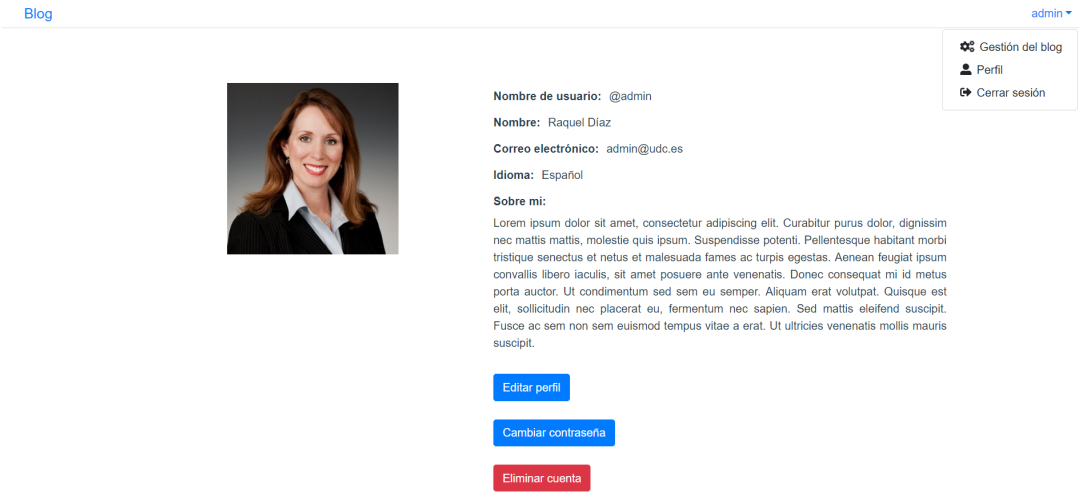


Figura 7.10: Página del perfil privado de un usuario

Figura 7.11: Página de gestión de artículos

un borrador, la fecha de última modificación, el usuario autor, el número de comentarios y las etiquetas. Además, se permite la modificación y borrado de cada artículo a través de los iconos que se muestran a su derecha, así como la eliminación de artículos de forma masiva mediante el botón situado en la parte superior de la lista, que elimina todos los artículos seleccionados.

Junto al botón de eliminar, se encuentra el botón que proporciona acceso al editor del blog para crear un nuevo artículo (figuras 7.12, 7.13 y 7.14). En la parte superior de la vista del editor puede verse un campo de texto para introducir el título del artículo y a continuación, los botones que permiten publicar y guardar el artículo, así como el botón para cancelar la edición. Bajo estos elementos se encuentra el editor propiamente dicho y a la derecha de este, el componente que permite añadir y guardar las etiquetas del artículo.

Haciendo uso de las pestañas situadas en la parte izquierda de la vista se puede acceder a la página de gestión de comentarios, que se muestra en la figura 7.15. En esta vista se presentan todos los comentarios realizados en los artículos del blog junto a su autor (si tiene) y a su fecha de creación. Además, se permite la eliminación de comentarios de forma individual a través del icono situado en su parte derecha o de forma masiva mediante el botón localizado en la parte superior de la lista, que elimina todos los comentarios seleccionados.

A través de las mencionadas pestañas también se puede acceder a la página de gestión de usuarios, que se muestra en la figura 7.16. En esta vista se muestran los usuarios registrados en el blog, indicando para cada uno de ellos su imagen de perfil, su nombre de usuario, su nombre y apellidos, su rol y su fecha de registro. Además, para todos los usuarios distintos del usuario autenticado se muestran dos iconos en la parte derecha que proporcionan acceso a la modificación y borrado del usuario en cuestión. Encima del listado de usuarios se encuentra un

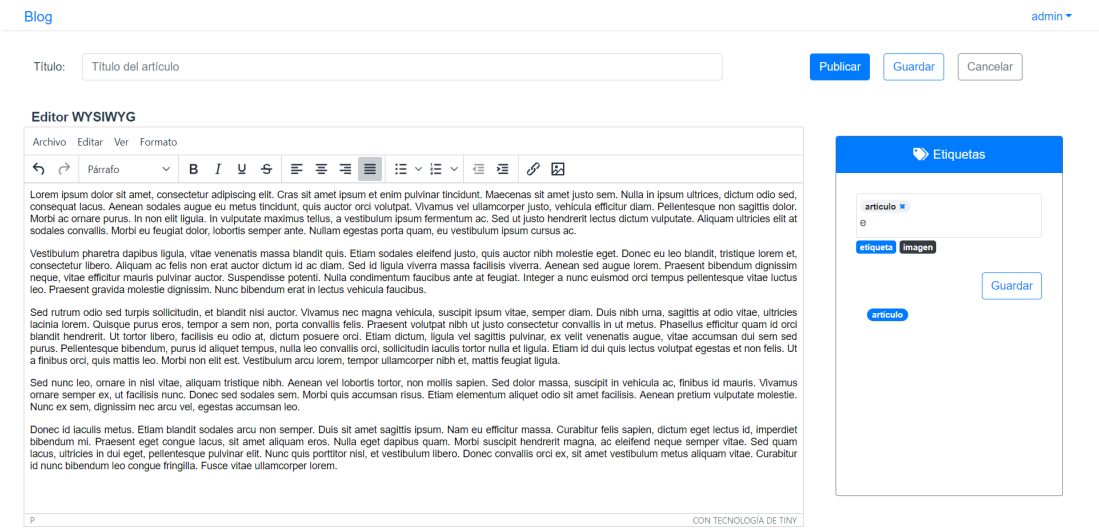


Figura 7.12: Página del editor WYSIWYG

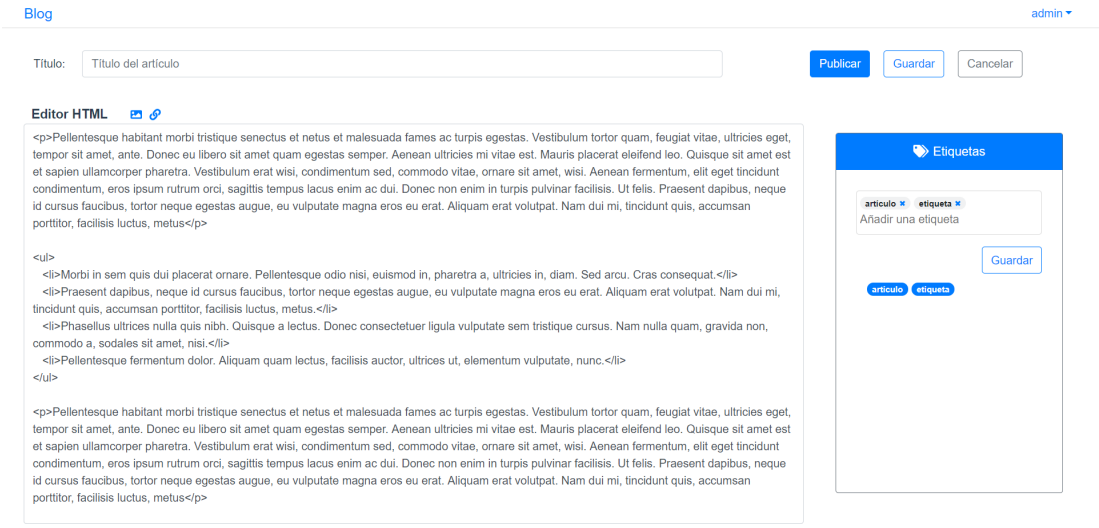


Figura 7.13: Página del editor HTML

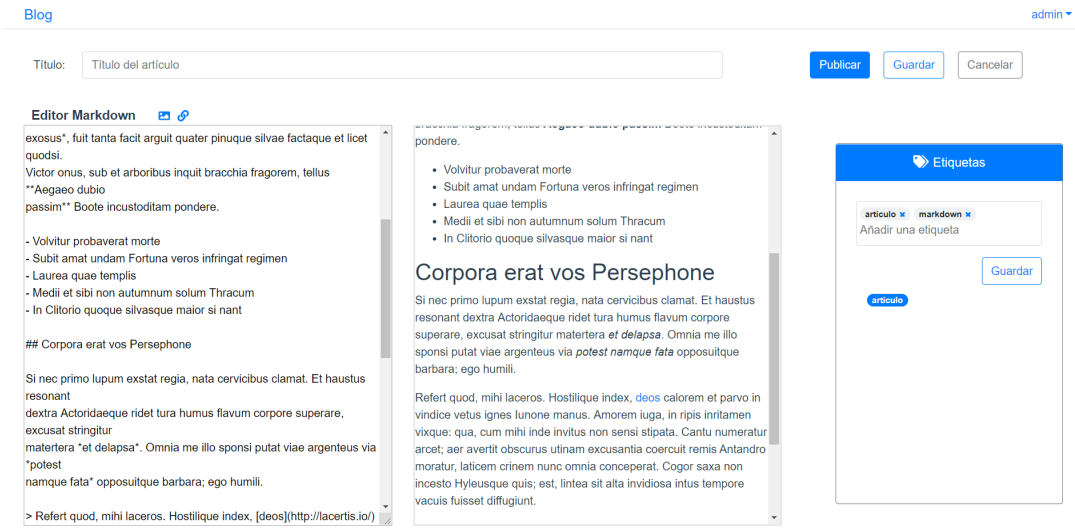


Figura 7.14: Página del editor Markdown

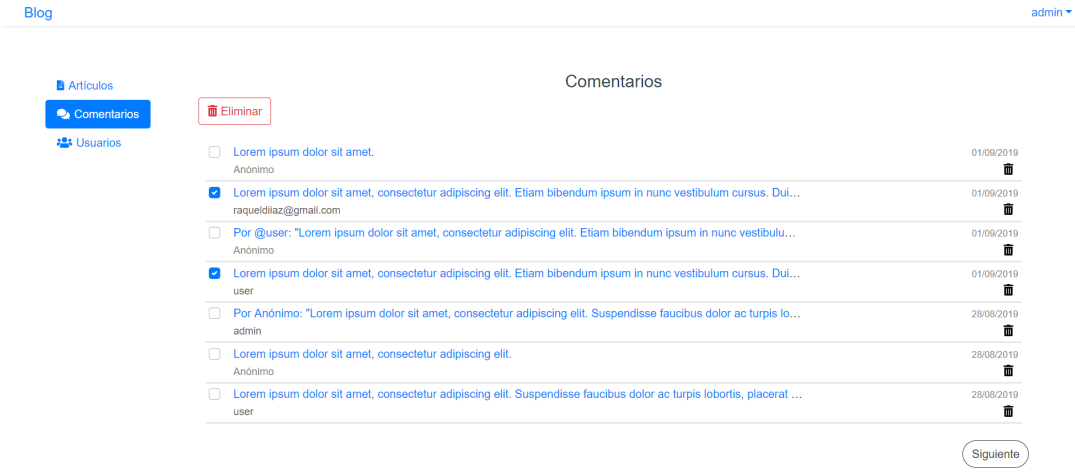


Figura 7.15: Página de gestión de comentarios

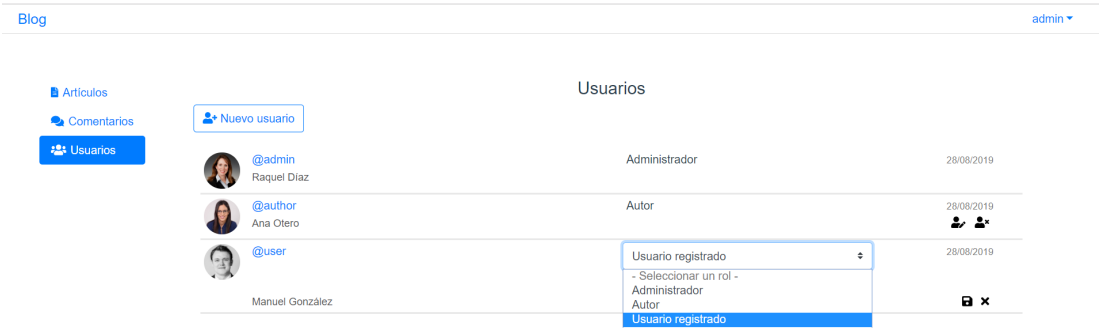


Figura 7.16: Página de gestión de usuarios

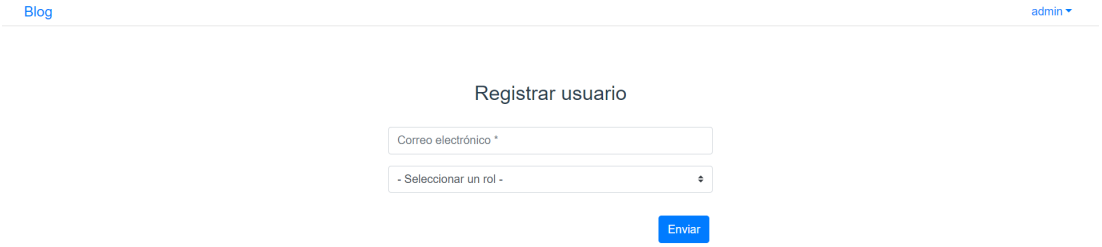


Figura 7.17: Página para añadir un nuevo usuario

botón que permite añadir un nuevo usuario al blog mediante la introducción de una dirección de correo electrónico y un rol (figura 7.17).

Cabe destacar que las figuras relacionadas con la de gestión del blog que se han mencionado, reflejan el contenido mostrado para un usuario con permisos de administrador.

Capítulo 8

Conclusiones y trabajo futuro

Al término de este proyecto, se puede afirmar que se han cumplido los objetivos establecidos para este:

- Se ha desarrollado una herramienta basada en líneas de producto software que permite generar aplicaciones web para publicar y gestionar un blog a partir de un conjunto de características, y que además cuenta con una interfaz sencilla que facilita el proceso de generación de productos.
- Se proporcionan productos generados seguros, de calidad y fáciles de utilizar gracias a la realización de múltiples pruebas, al empleo de un sistema de seguridad y al desarrollo de una interfaz web simple e intuitiva.
- Se proporcionan las instrucciones necesarias para la configuración y despliegue de los productos generados, facilitando la instalación de los mismos.

Durante el proceso de desarrollo del proyecto, se han podido aplicar multitud de conocimientos adquiridos a lo largo de toda la titulación, entre los que se pueden mencionar:

- **Conocimientos técnicos** relacionados con las tecnologías y herramientas empleadas en este proyecto y que han permitido llevar a cabo el desarrollo de todas las tareas planificadas.
- Conocimientos sobre la aplicación de **metodologías ágiles**, especialmente Scrum, y sobre el procedimiento a seguir a la hora de planificar y estimar las tareas de un proyecto.

Además, gracias a la realización de este trabajo se han adquirido multitud de conocimientos nuevos. Entre ellos se pueden destacar:

- Se han comprendido los pilares básicos de las diferentes técnicas existentes para la implementación de líneas de producto software y se ha podido conocer de primera mano el procedimiento a seguir a la hora de desarrollar una LPS basada en la técnica de *scaffolding*.
- Se ha profundizado en el desarrollo de aplicaciones del lado cliente, mediante el uso de Vue.js y de multitud de bibliotecas JavaScript.
- Se han completado los conocimientos ya existentes sobre el uso de las tecnologías empleadas en el desarrollo del servidor web, prestando más atención a la configuración del proyecto y a la realización de pruebas.
- Se ha experimentado de primera mano lo que supone llevar a cabo un proyecto de este alcance, resaltando la importancia de la planificación y el seguimiento constante del desarrollo.

Pese a todo esto, la solución desarrollada podría mejorarse en algunos aspectos:

- **Despliegue automático de los productos generados** mediante el empleo de algún servidor de aplicaciones online como Heroku, de forma que las aplicaciones web generadas fuesen accesibles desde internet.
- **Actualización de las características de un producto** generado previamente manteniendo los cambios locales realizados y sin necesidad de generar un nuevo producto.

Apéndices

Apéndice A

Diagrama del modelo de características

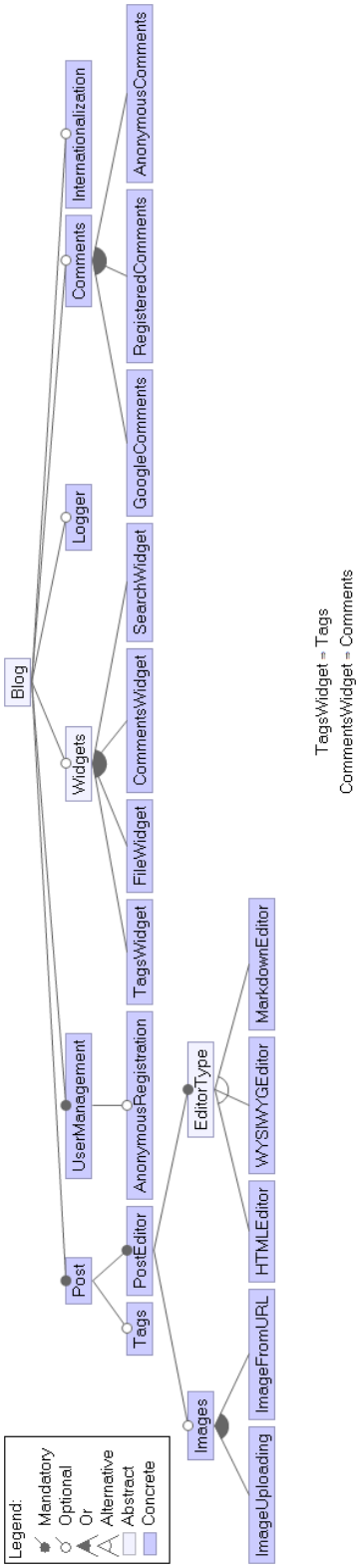


Figura A.1: Modelo de características de la LPS

Apéndice B

Prototipos de pantallas

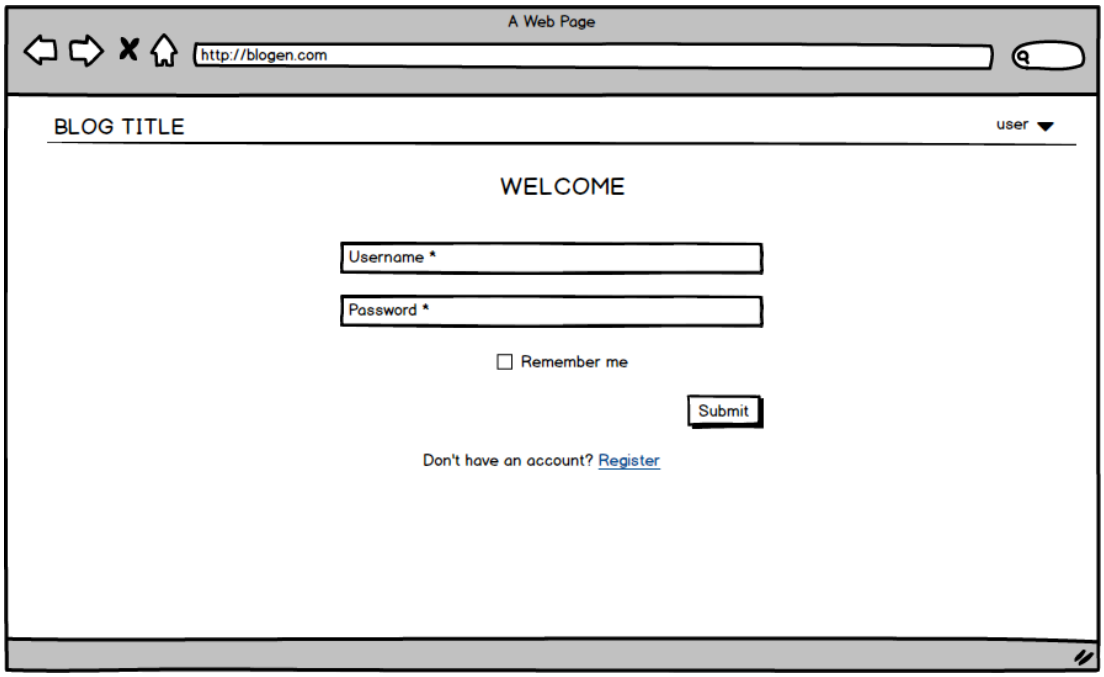


Figura B.1: Página de inicio de sesión

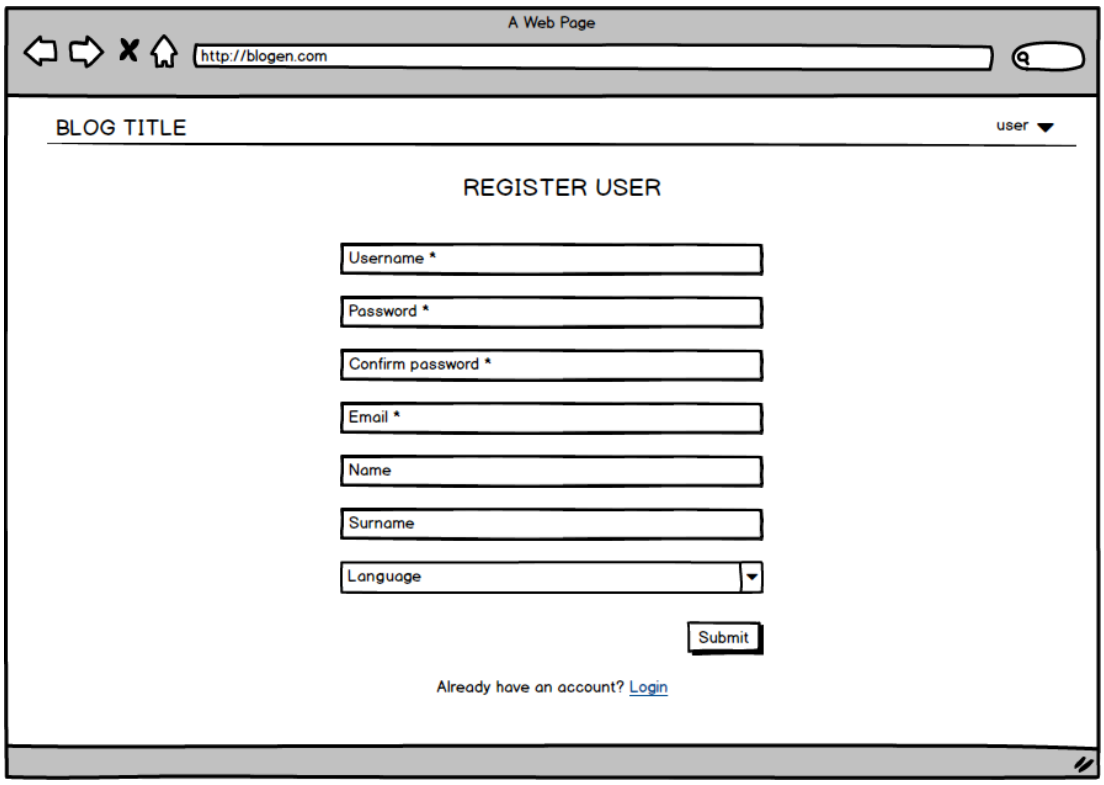


Figura B.2: Página de registro

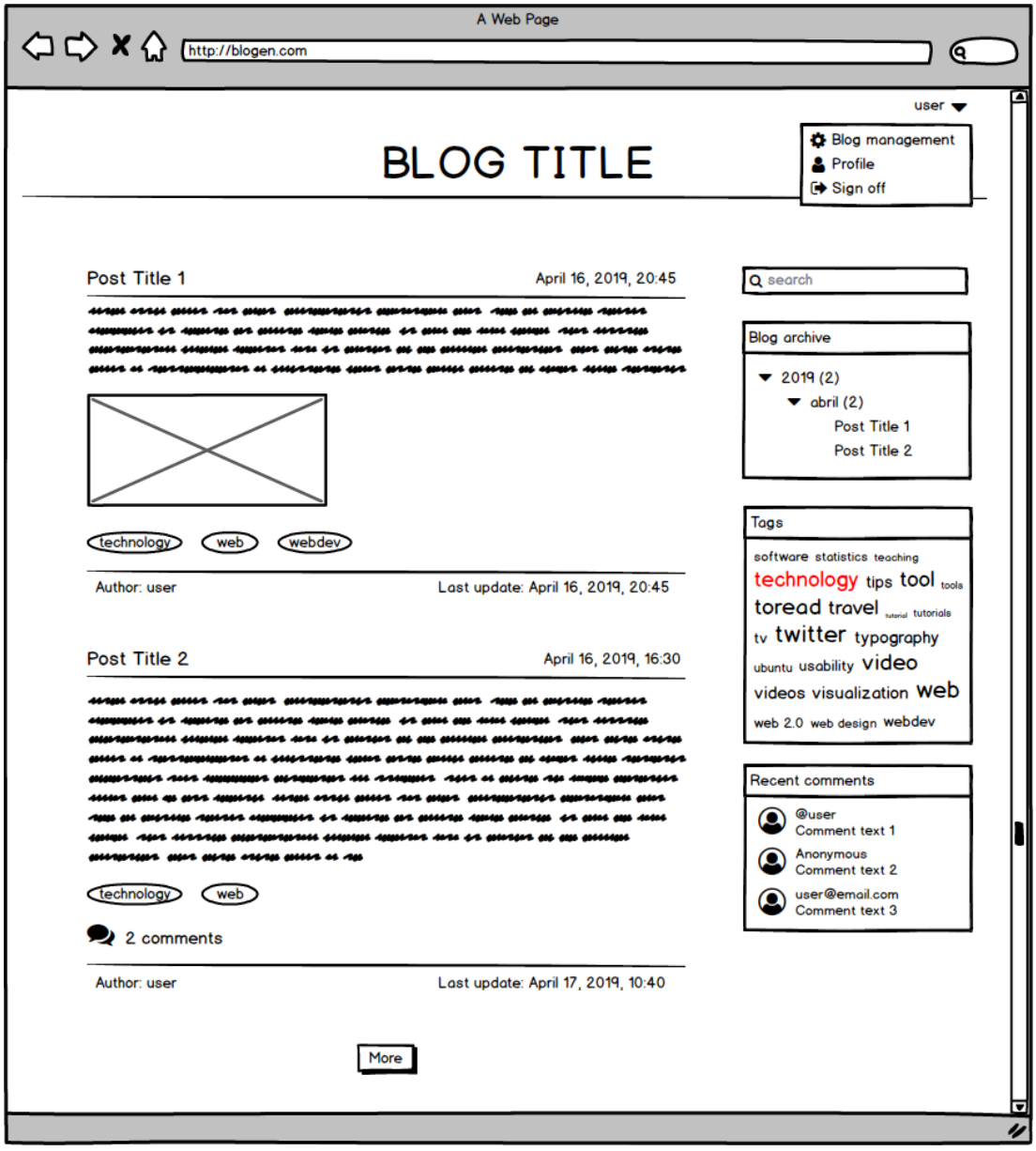


Figura B.3: Página principal con cabecera de usuario autenticado

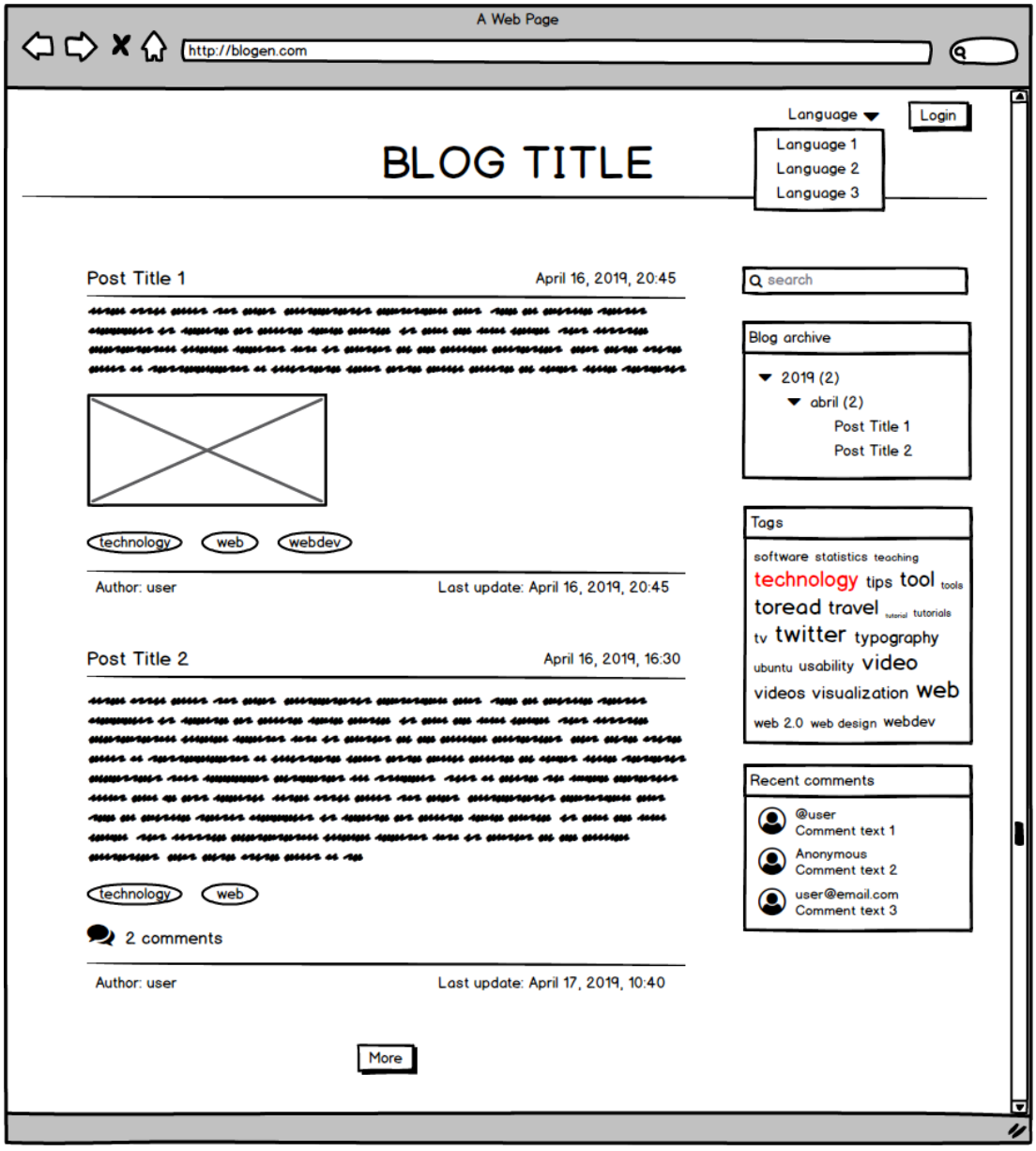


Figura B.4: Página principal con cabecera de usuario no autenticado

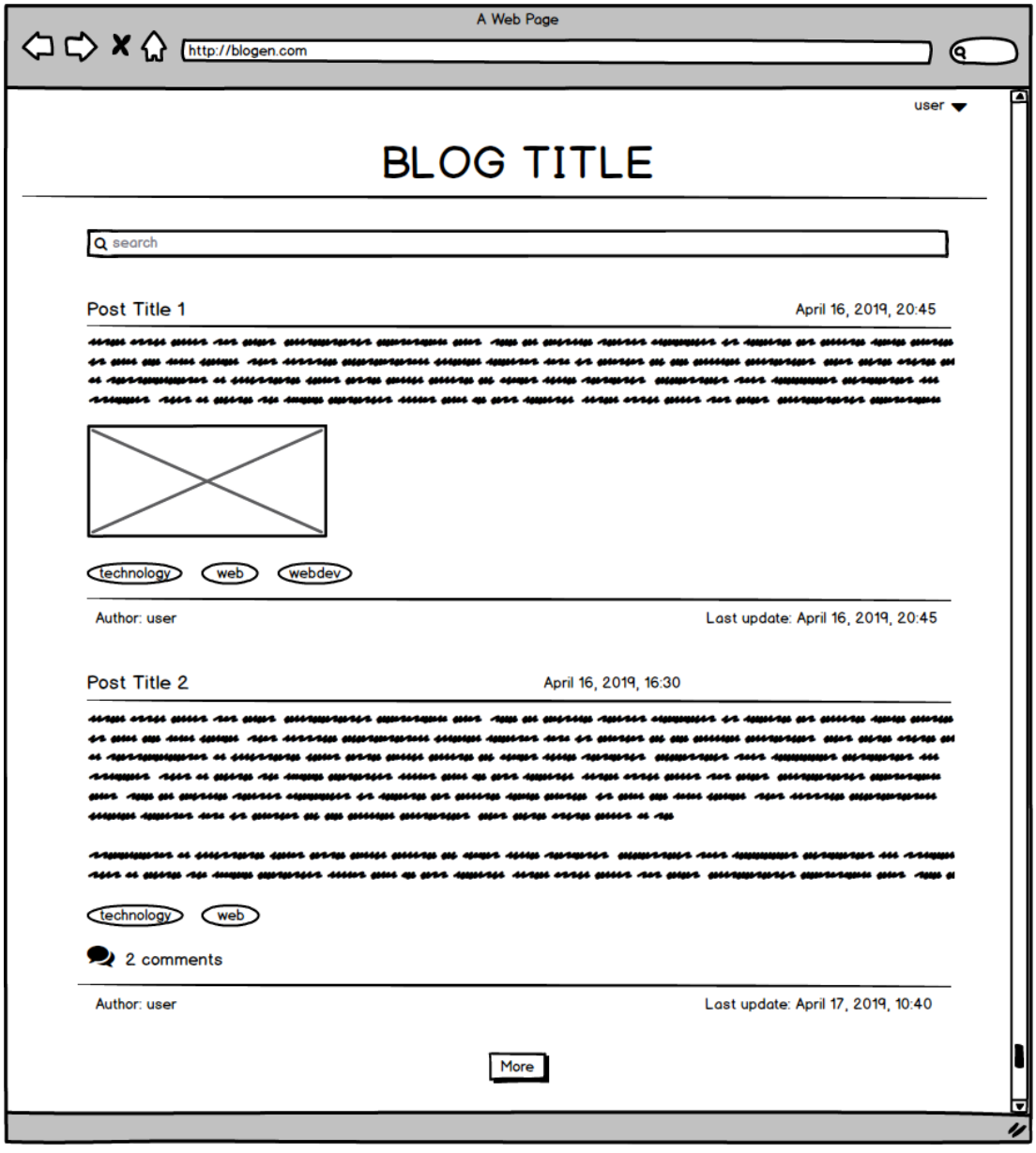


Figura B.5: Página principal con barra de búsqueda centrada

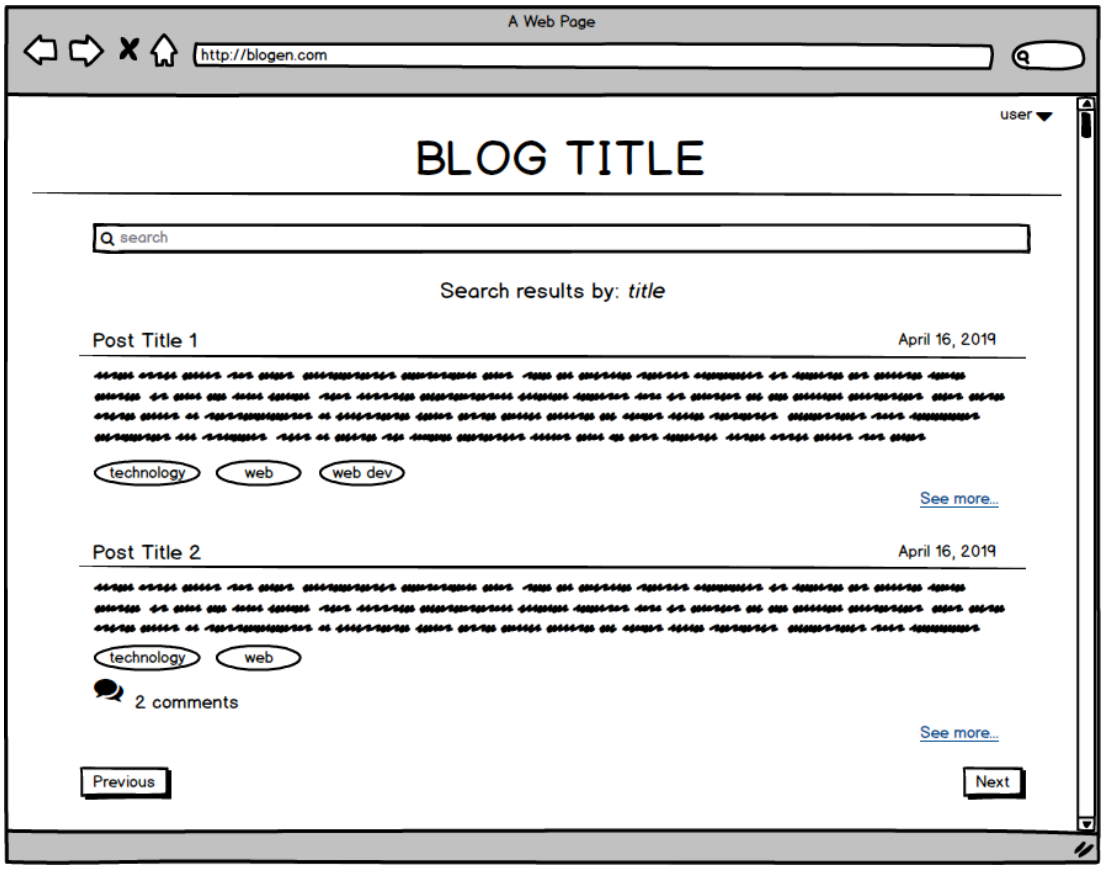


Figura B.6: Página de resultados de la búsqueda por palabras clave

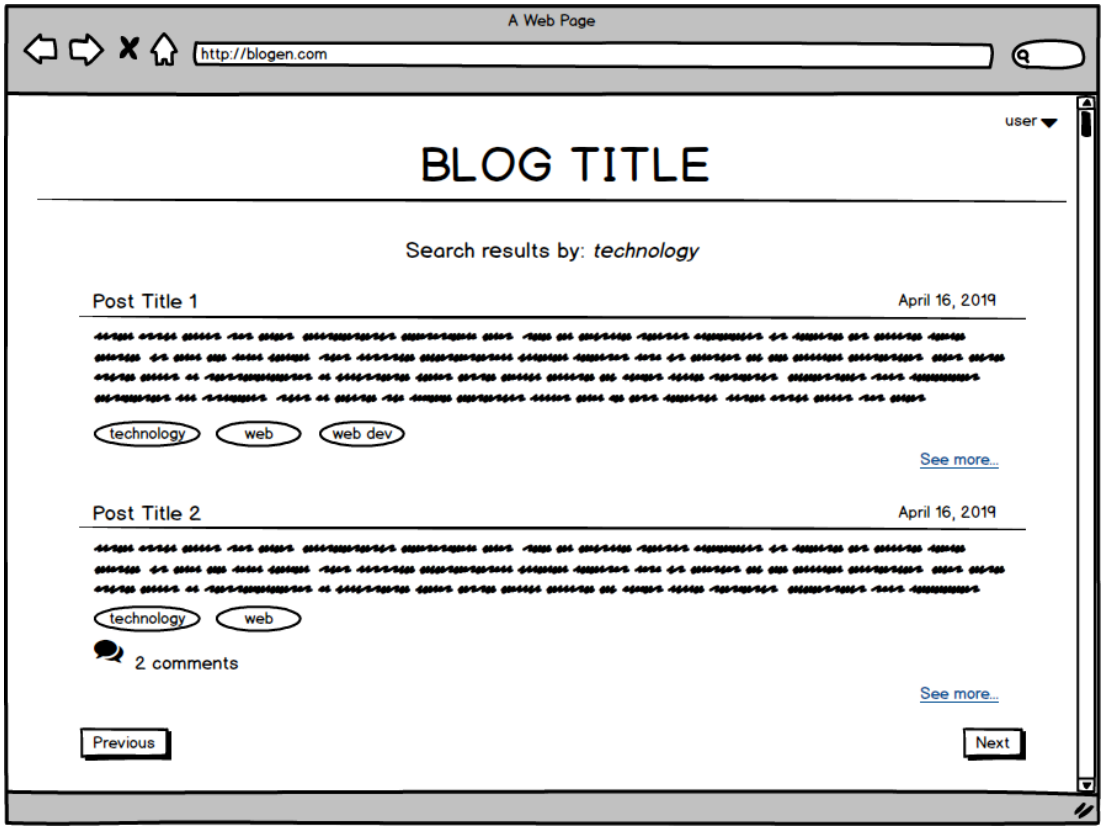


Figura B.7: Página de resultados de la búsqueda por etiqueta

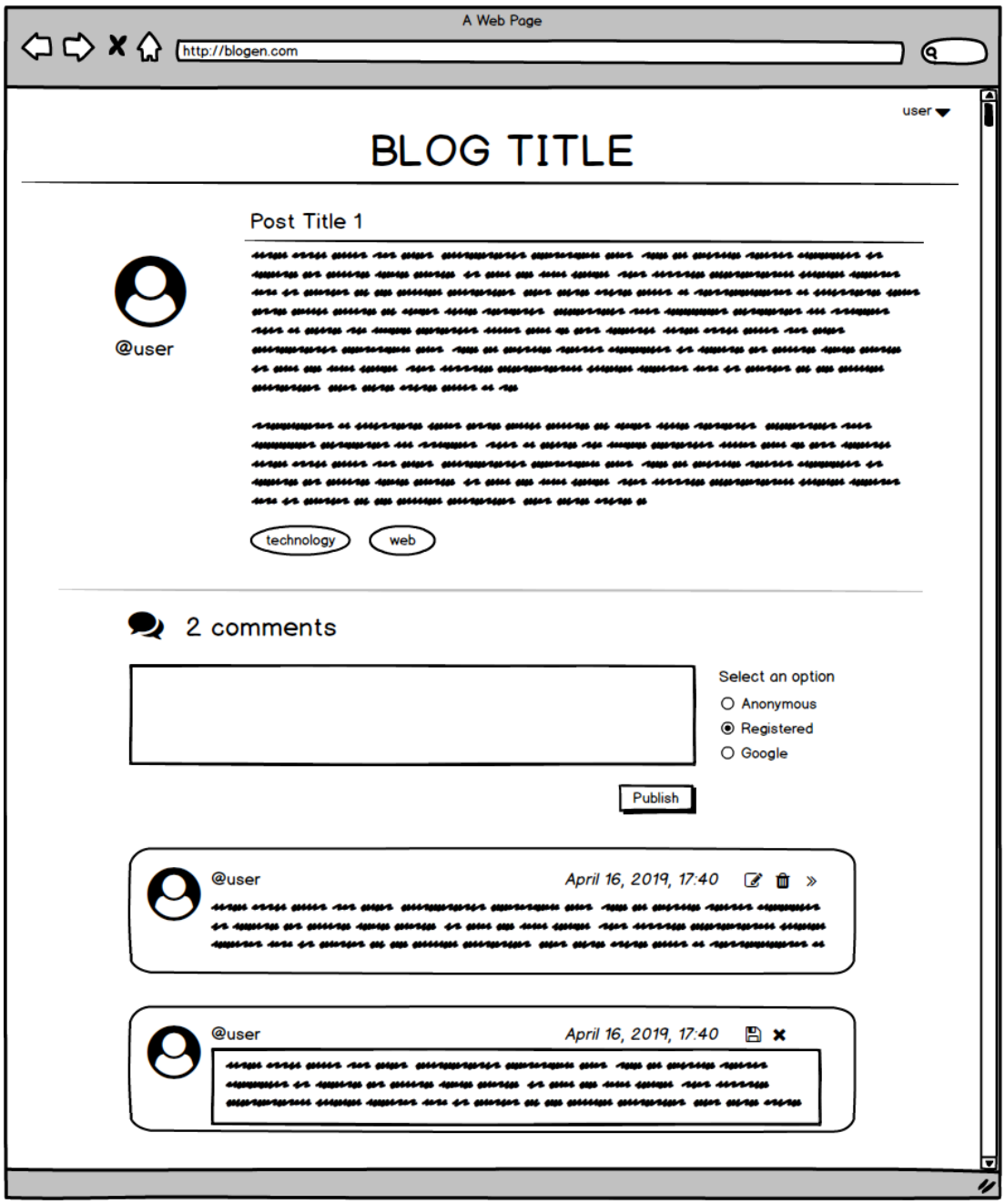


Figura B.8: Página de un artículo

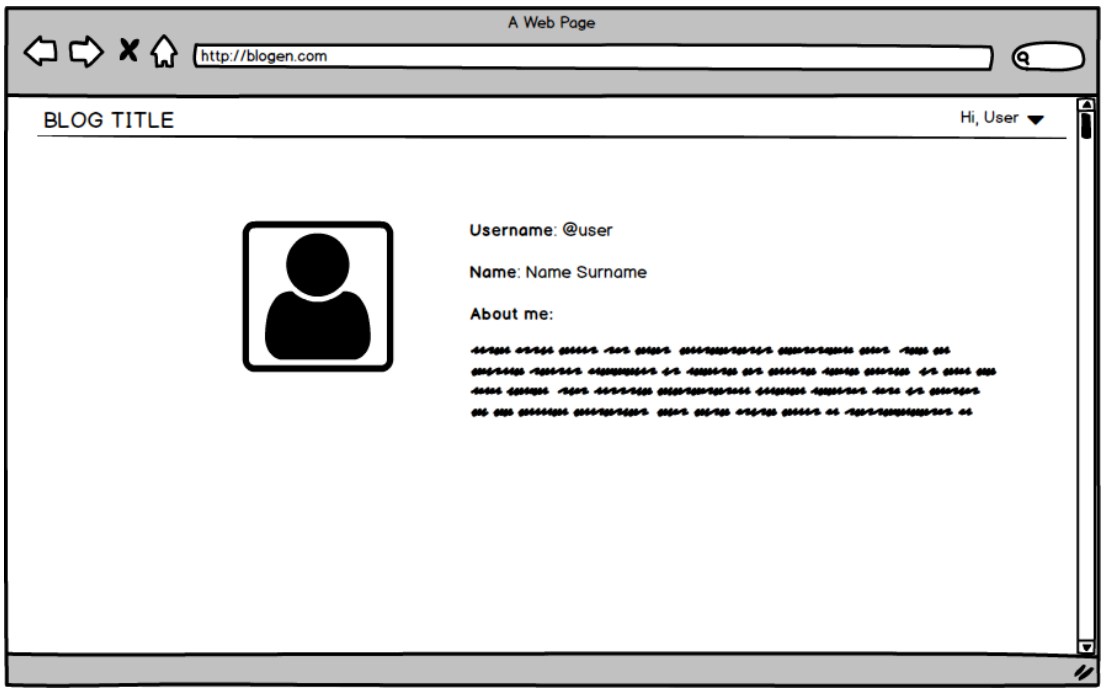


Figura B.9: Perfil público de un usuario

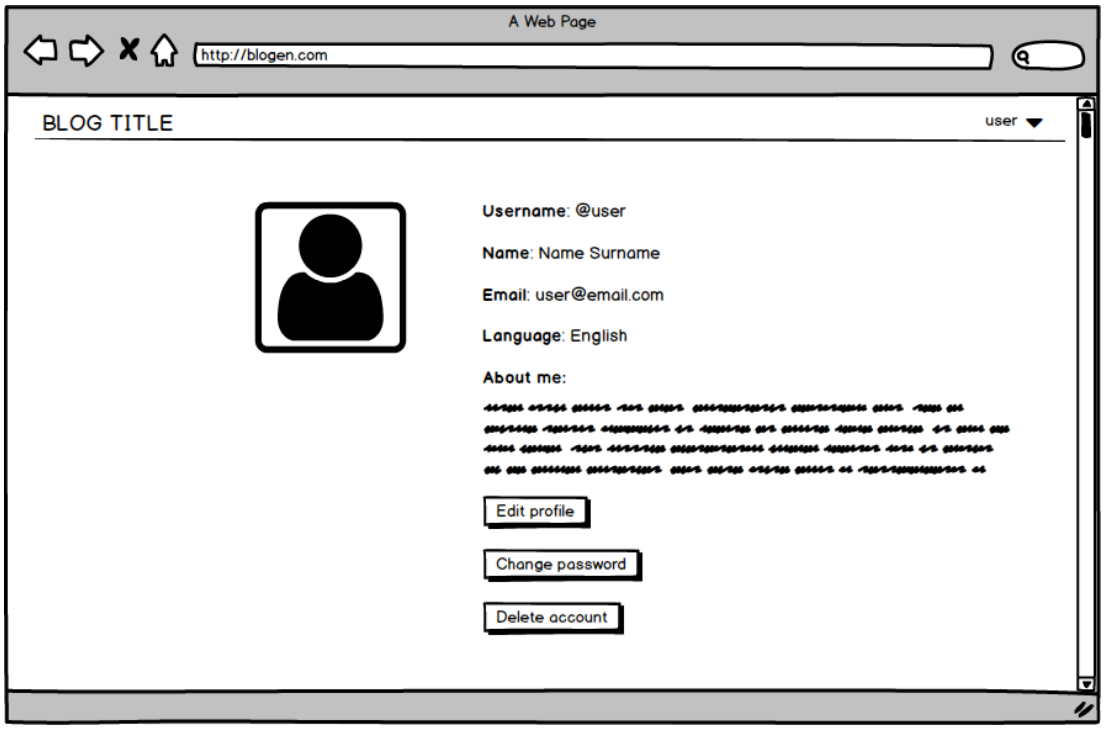


Figura B.10: Perfil privado de un usuario

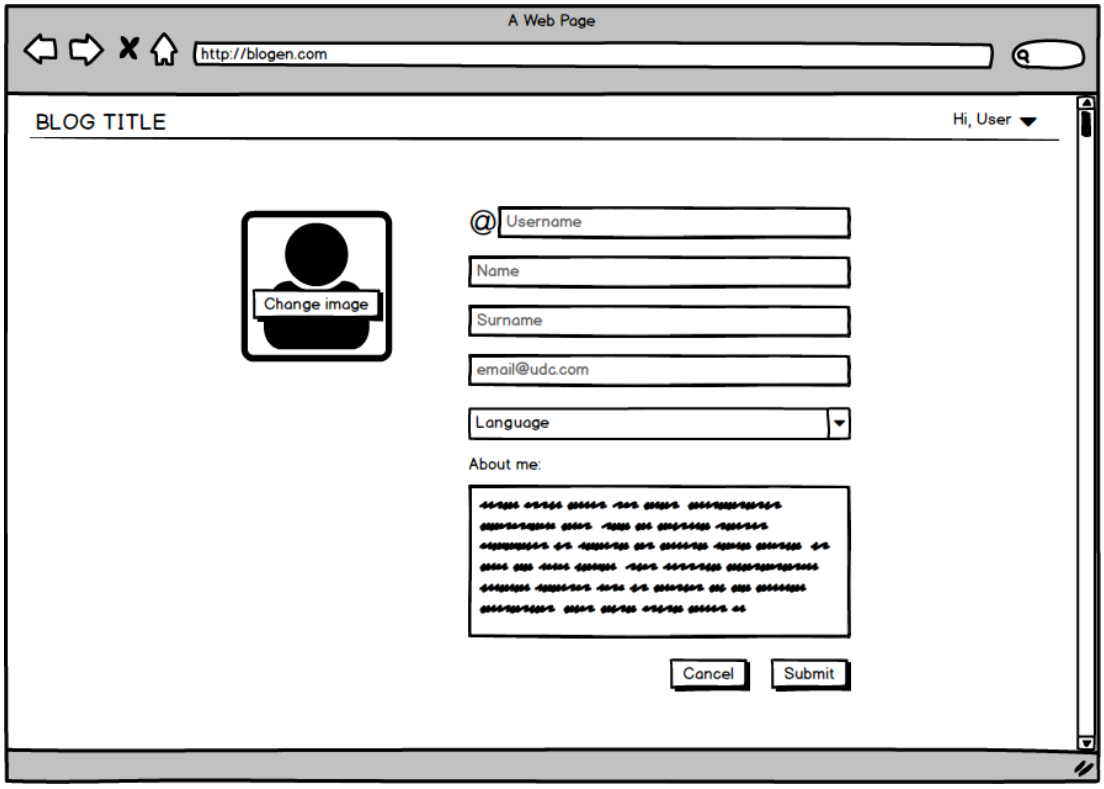


Figura B.11: Página de modificación del perfil de usuario

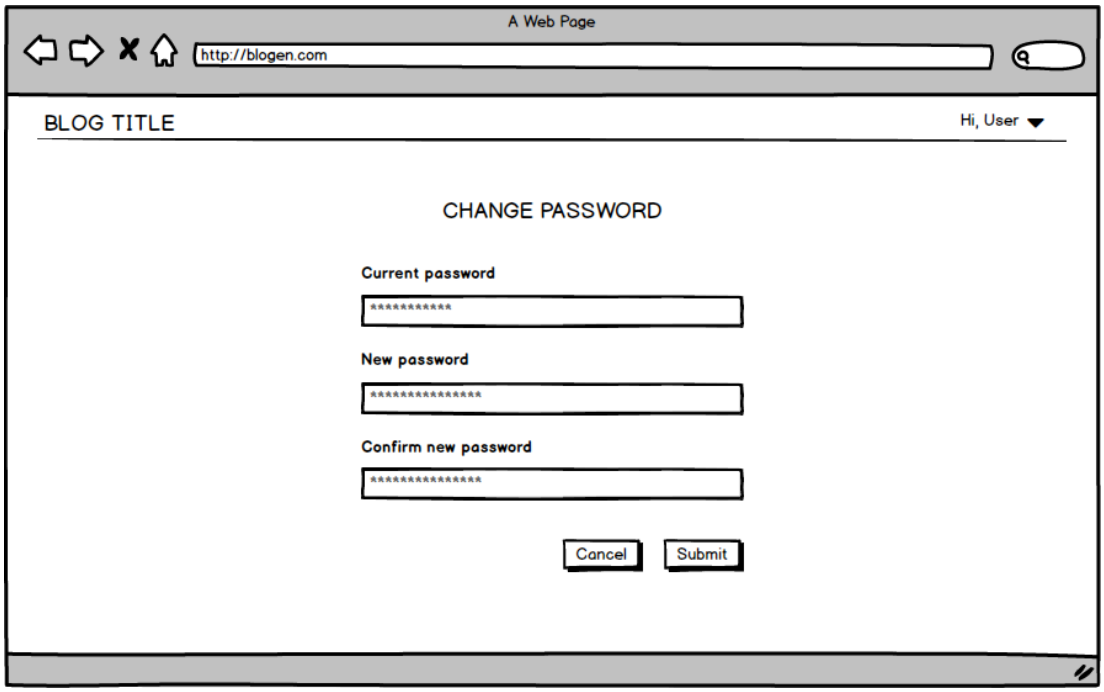


Figura B.12: Página de modificación de contraseña

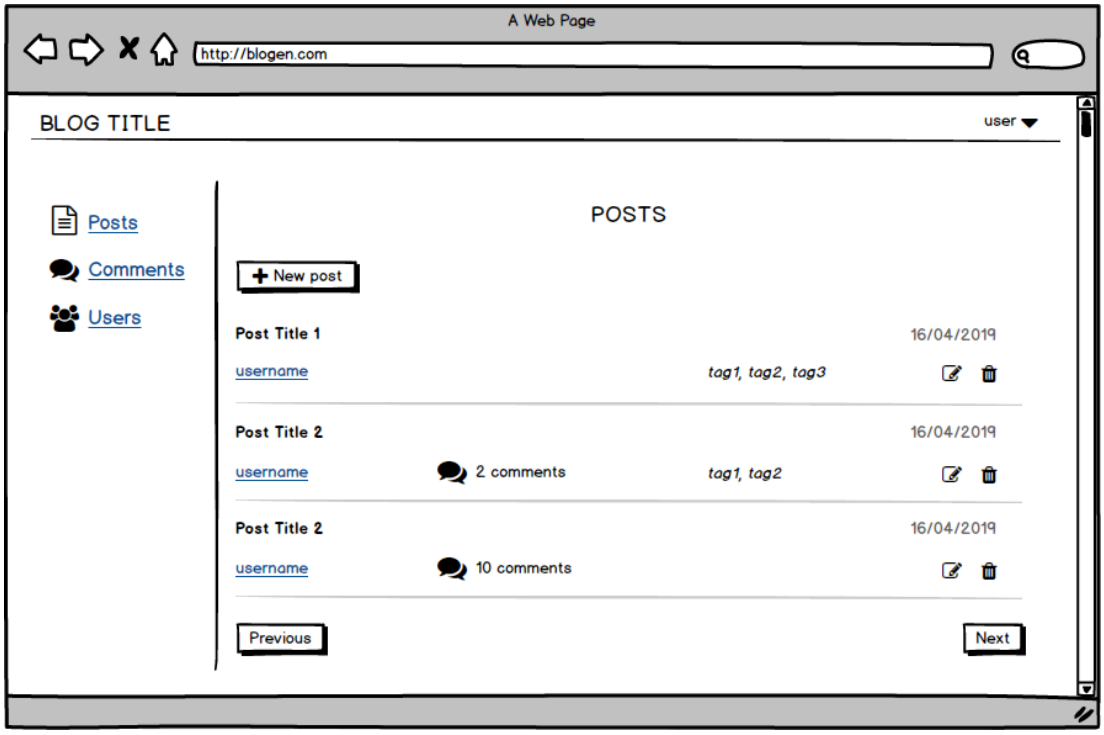


Figura B.13: Página de gestión de artículos

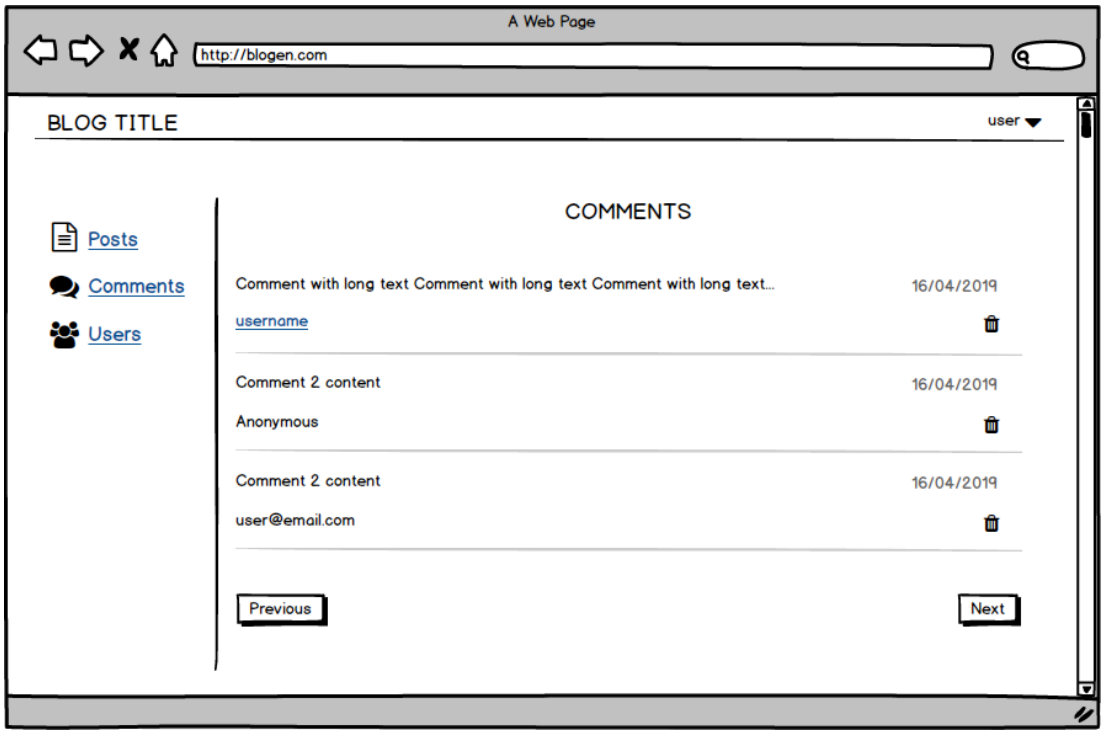


Figura B.14: Página de gestión de comentarios

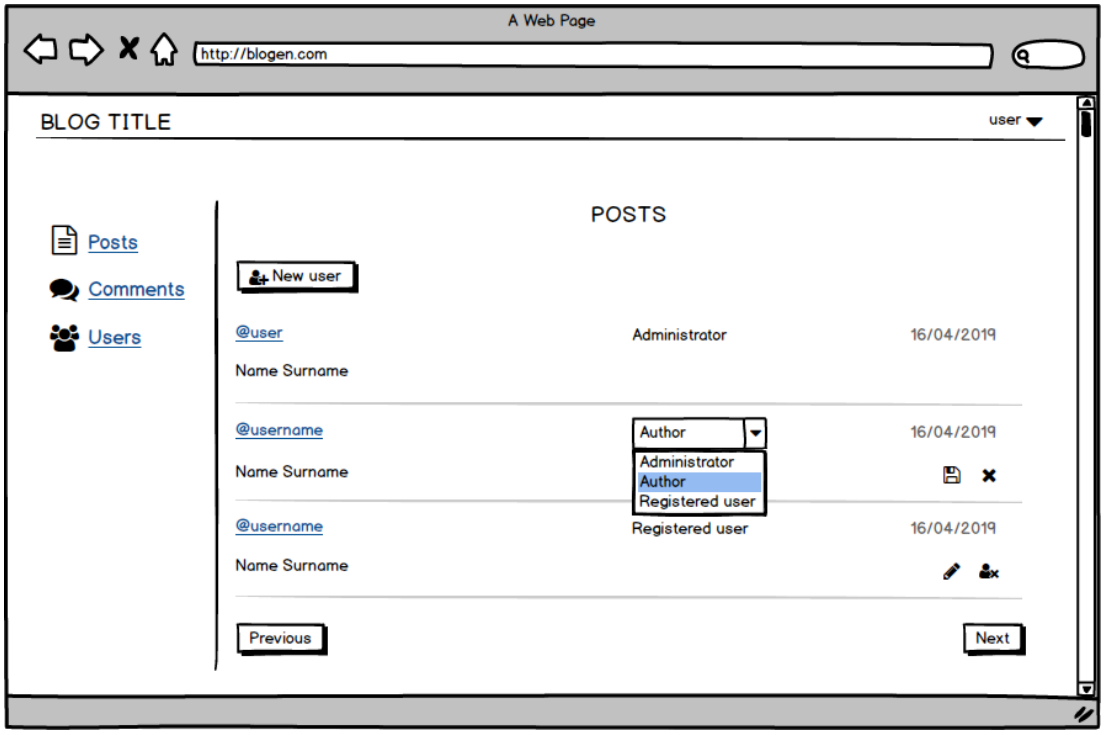


Figura B.15: Página de gestión de usuarios

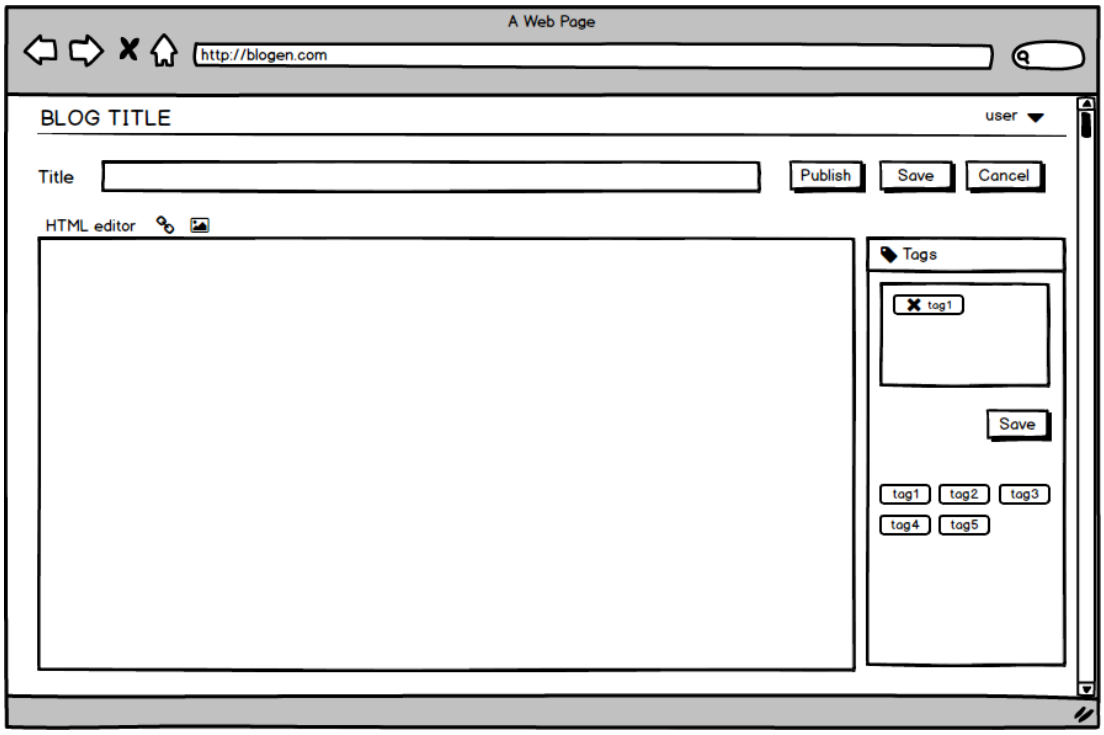


Figura B.16: Página del editor HTML

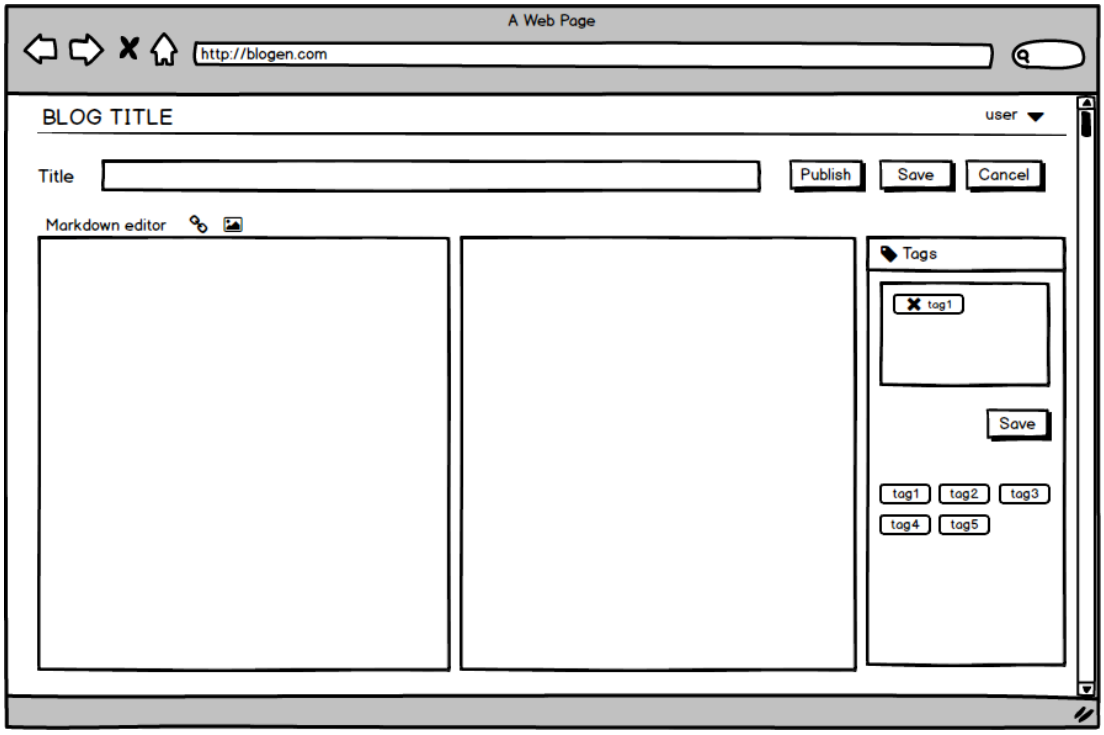


Figura B.17: Página del editor Markdown

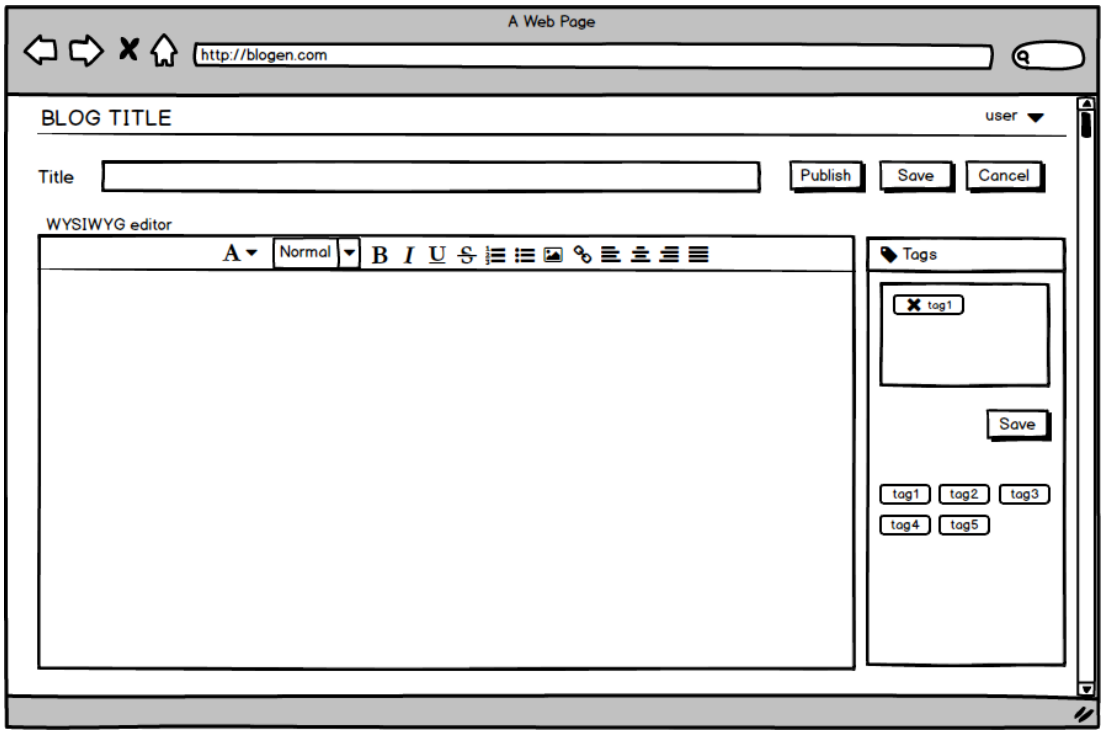


Figura B.18: Página del editor WYSIWYG

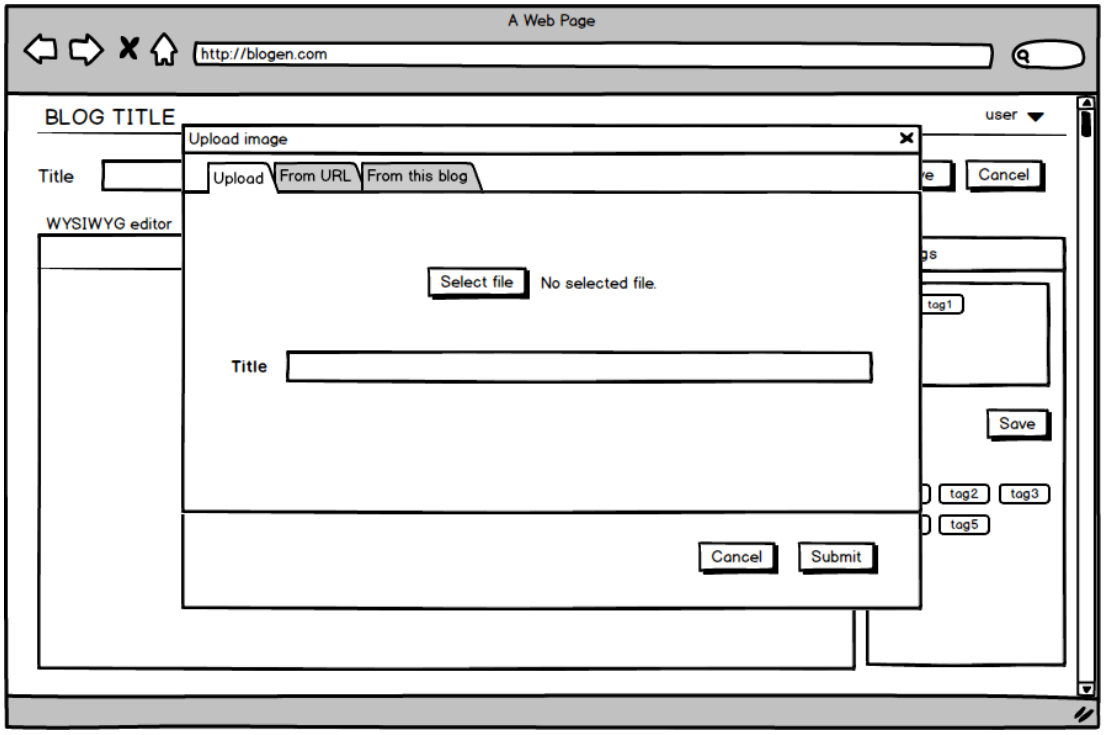


Figura B.19: Ventana de añadir imagen desde archivo

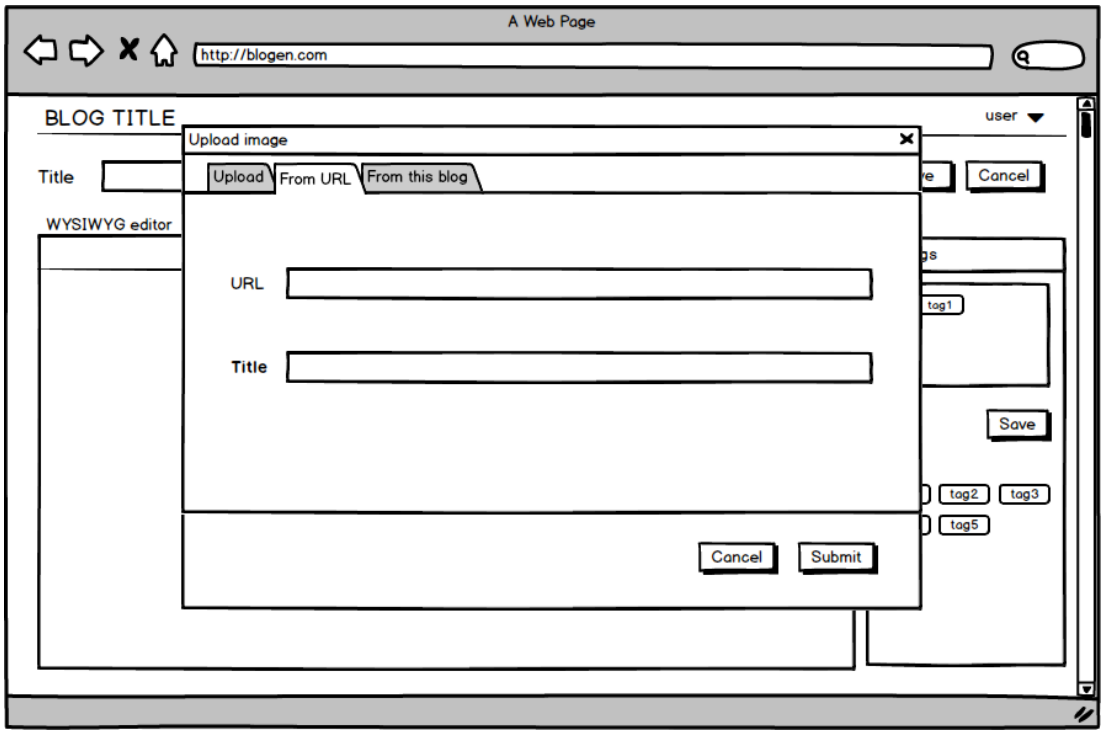


Figura B.20: Ventana de añadir imagen desde url

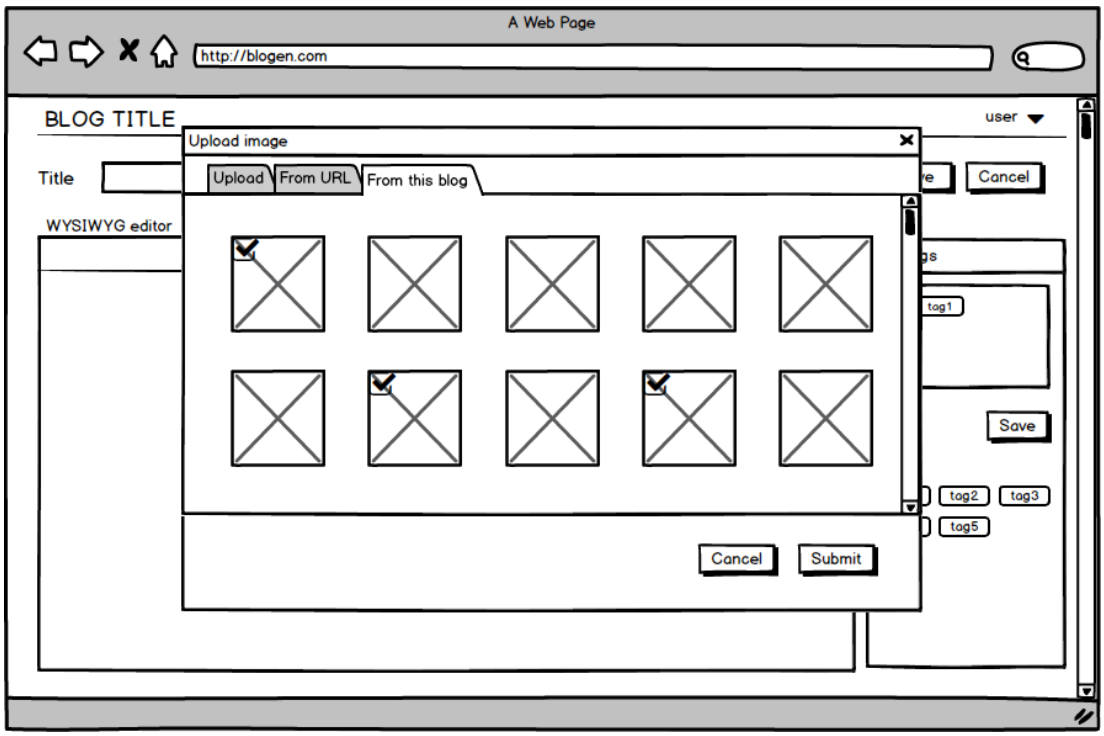


Figura B.21: Ventana de añadir imagen desde galería

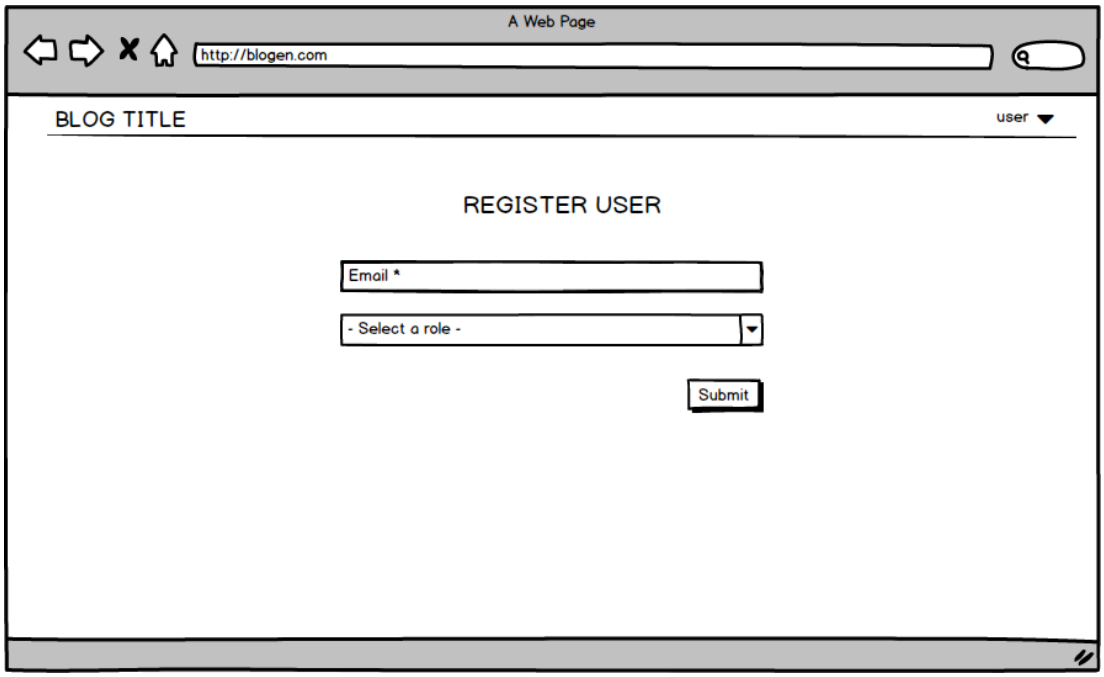


Figura B.22: Página de añadir un usuario

Apéndice C

Correspondencias

C.1 API REST

En esta sección se muestran las correspondencias entre las operaciones expuestas por la API REST y los métodos de los controladores encargados de gestionar las peticiones. Para facilitar la lectura se utilizarán las siguientes abreviaturas: **U** para referirse a UserProfileController, **P** para referirse a PostController, **I** para referirse a ImageController, **C** para referirse a CommentController y **T** para referirse a TagController.

1. **POST /login**: -
2. **POST /user/register**: U.registerUser
3. **POST /user/save**: U.saveUser
4. **POST /user/savePassword**: U.savePassword
5. **POST /user/saveImage**: U.saveProfileImage
6. **DELETE /user/delete**: U.deleteUser
7. **POST /user/registerUser**: U.registerUserAdmin
8. **PUT /user/saveRole**: U.saveUserRole
9. **GET /user/findAll**: U.findAll
10. **GET /user/find**: U.findUser
11. **GET /user/images/{id}**: U.getImage
12. **POST /post/save**: P.savePost

13. **POST /post/publish**: P.publishPost
14. **DELETE /post/delete**: P.deletePost
15. **GET /post/getAll**: P.findAll
16. **GET /post/find**: P.findPost
17. **GET /post/findByTag**: P.findByTag
18. **GET /post/findPublished**: P.findPublished
19. **GET /post/findYears**: P.findYears
20. **GET /post/findMonths**: P.findMonths
21. **GET /post/findByMonth**: P.findByMonthOfYear
22. **GET /post/findByKeywords**: P.findByKeywords
23. **POST /image/save**: I.saveImage
24. **POST /image/saveURL**: I.saveImageURL
25. **GET /image/getAll**: I.getAll
26. **GET /image/uploads/{id}**: I.getImage
27. **POST /comment/save**: C.saveComment
28. **POST /comment/saveAnonymous**: C.saveComment
29. **POST /comment/saveGoogle**: C.saveComment
30. **DELETE /comment/delete**: C.deleteComment
31. **DELETE /comment/deleteOnPost**: C.deleteCommentOnPost
32. **DELETE /comment/deleteGoogle**: C.deleteComment
33. **GET /comment/findAll**: C.findAll
34. **GET /comment/findByPost**: C.findByPost
35. **GET /comment/getNumberByPost**: C.getNumberByPost
36. **GET /comment/findRecents**: C.findRecents
37. **POST /tag/saveAll**: T.saveAll

Historia	Petición(es)
1	1
2	-
3	12, 13, 23, 24, 25, 26
4	15, 35
5	15, 35
6	12, 13, 23, 24, 25, 26
7	12, 13, 23, 24, 25, 26
8	14
9	14
10	16, 26, 11
11	38, 37, 12
12	38, 37, 12
13	38, 37, 12
14	38, 37, 12
15	16
16	17, 35
17	34, 11
18	28
19	27
20	27
21	-
22	-
23	31
24	33
25	33

Historia	Petición(es)
26	30
27	30
28	2
29	10, 11
30	3, 5
31	4
32	10, 11
33	6
34	9, 11
35	6
36	8
37	7
38	18, 26, 35
39	19, 20, 21
40	36, 11
41	39
42	22, 35
43	3
44	-
45	-
46	29
47	29
48	32
49	-

Tabla C.1: Correspondencia entre las historias de usuario y las peticiones HTTP

38. **GET /tag/getAll**: T.getAll

39. **GET /tag/getCloud**: T.getCloud

C.2 Tabla de correspondencias

En la tabla C.1 se muestran las correspondencias entre las historias de usuario definidas para el producto y las peticiones al servidor web necesarias para llevarlas a cabo.

Apéndice D

Modelo de características en formato XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<featureModel>
    <properties/>
    <struct>
        <and abstract="true" mandatory="true" name="Blog">
            <and mandatory="true" name="Post">
                <feature name="Tags"/>
                <and mandatory="true" name="PostEditor">
                    <or name="Images">
                        <feature name="ImageUploading"/>
                        <feature name="ImageFromURL"/>
                    </or>
                    <alt abstract="true" mandatory="true"
    name="EditorType">
                        <feature name="HTMLEditor"/>
                        <feature name="WYSIWYGEditor"/>
                        <feature name="MarkdownEditor"/>
                    </alt>
                </and>
            </and>
            <and mandatory="true" name="UserManagement">
                <feature name="AnonymousRegistration"/>
            </and>
            <or abstract="true" name="Widgets">
                <feature name="TagsWidget"/>
                <feature name="FileWidget"/>
                <feature name="CommentsWidget"/>
                <feature name="SearchWidget"/>
            </or>
            <feature name="Logger"/>
```

```
            <or name="Comments">
                <feature name="GoogleComments"/>
                <feature name="RegisteredComments"/>
                <feature name="AnonymousComments"/>
            </or>
            <feature name="Internationalization"/>
        </and>
    </struct>
    <constraints>
        <rule>
            <imp>
                <var>TagsWidget</var>
                <var>Tags</var>
            </imp>
        </rule>
        <rule>
            <imp>
                <var>CommentsWidget</var>
                <var>Comments</var>
            </imp>
        </rule>
    </constraints>
    <calculations Auto="true" Constraints="true" Features="true"
    Redundant="true" Tautology="true"/>
    <comments/>
    <featureOrder userDefined="false"/>
</featureModel>
```

Apéndice E

Manual de instalación

E.1 Herramienta de generación

Para permitir la instalación y despliegue de la herramienta es necesario tener instalado en el sistema Node.js versión 10.15. El despliegue de la aplicación se realiza mediante la ejecución de los siguientes comandos:

```
npm install
npm run serve
```

Una vez desplegada la aplicación se podrá acceder a ella a través de: http://localhost:8082

E.2 Producto generado

Los **requisitos del sistema** para la instalación y despliegue de los productos generados son:

- Java 8
- Apache Maven versión 3.6.1
- Servidor de PostgreSQL versión 11.3
- Node.js versión 10.15

En cuanto la **configuración del entorno**, es necesario llevar a cabo las siguientes tareas en el proyecto del servidor de la aplicación (*blogen-rest-service*):

- Crear una base de datos en el servidor de PostgreSQL y configurar los parámetros de la figura E.1 en el fichero /src/main/resources/application.properties. Se deberán sustituir los elementos resaltados por los valores correspondientes de la base de datos creada.

```
spring.datasource.url=jdbc:postgresql://localhost:5432/databaseName?useUnicode=yes&characterEncoding=UTF-8
spring.datasource.username=databaseUser
spring.datasource.password=databasePassword
```

Figura E.1: Parámetros de configuración de la base de datos

- Establecer en el parámetro *imagesFolder* del fichero /src/main/resources/config.properties la ruta a la carpeta del sistema en la que se desean almacenar las imágenes de la aplicación.

Para el despliegue de la aplicación se distingue entre el despliegue del servidor web y el despliegue del cliente web.

Por una parte, para el **despliegue del servidor** basta con ejecutar el siguiente comando en el directorio raíz de su proyecto (*blogen-rest-service*):

```
1    mvn spring-boot:run
```

Por otra parte, para el **despliegue del cliente** es necesario ejecutar los siguientes comandos en el directorio raíz de su proyecto (*blogen-web-client*):

```
1    npm install
2    npm run serve
```

Una vez desplegada la aplicación se podrá acceder a ella a través de: http://localhost:8081

Apéndice F

Contenido del CD

El CD incluido contiene los siguientes ficheros:

- **DiazOtero_Raquel_TFG_2019.pdf**: Memoria del trabajo de fin de grado en formato PDF.
- **DiazOtero_Raquel_TFG_2019_resumo.pdf**: Resumen del trabajo de fin de grado en formato PDF.
- **DiazOtero_Raquel_TFG_2019_anexo.zip**: Código fuente de la herramienta de generación (*blogen-web-generator*).

Lista de acrónimos

API *Application Program Interface.*

CLI *Command Line Interface.*

CRUD *Create-Read-Update-Delete.*

CSS *Cascading Style Sheets.*

DAO *Data Access Object.*

DOM *Document Object Model.*

DTO *Data Transfer Object.*

FOP *Feature-Oriented Programming.*

HTML *HyperText Markup Language.*

HTTP *Hypertext Transfer Protocol.*

IDE *Integrated Development Environment.*

Java EE *Java Enterprise Edition.*

JSON *JavaScript Object Notation.*

JPA *Java Persistence API.*

JWT *JSON Web Token.*

LPS *Línea de Producto Software.*

NPM *Node Package Manager.*

REST *Representational State Transfer.*

SPA *Single Page Application.*

SPL *Software Product Line.*

SQL *Standardized Query Language*

URL *Uniform Resource Locator.*

WYSIWYG *What You See Is What You Get.*

XML *Extensible Markup Language.*

Glosario

Back-end Parte de un sistema informático que no es accesible por el usuario y que se encarga del acceso y manipulación de los datos.

Bug Error o problema en un programa informático.

Framework Entorno software reusable que proporciona una funcionalidad particular para facilitar el desarrollo de una aplicación informática.

Front-end Parte de un sistema informático que es accesible por el usuario y que se encarga de mostrar la información y gestionar las interacciones del usuario.

Issue unidad de trabajo para completar una mejora en un sistema.

Línea de producto software Conjunto de sistemas software, que comparten un conjunto común de características y que se desarrollan a partir de un sistema común de activos base (componentes reutilizables) de una manera preestablecida.

Log Sistema de registro de la actividad realizada por un programa informático.

Scaffolding Técnica que consiste en la generación de código a partir de plantillas predefinidas y de una especificación proporcionada por el desarrollador.

Plugin componente software que agrega una característica específica a un programa informático.

Programación orientada a características Paradigma de programación para la construcción de sistemas software a gran escala que se basa en la descomposición del sistema en términos de las características que proporciona.

Bibliografía

[1] "Ahead tool suite." [Online]. Available: https://www.cs.utexas.edu/~schwartz/ATS/fopdocs/

[2] "Featurehouse: Language-independent, automated software composition." [Online]. Available: https://www.infosun.fim.uni-passau.de/spl/apel/fh/

[3] "Página web de yeoman." [Online]. Available: https://yeoman.io/

[4] "Cide: Virtual separation of concerns." [Online]. Available: https://ckaestne.github.io/CIDE/

[5] "Página web de wordpress.com." [En línea]. Disponible en: https://wordpress.com/

[6] "Página web de blogger." [En línea]. Disponible en: https://www.blogger.com/

[7] "Página web de wix." [En línea]. Disponible en: https://www.wix.com

[8] "Página web de medium." [En línea]. Disponible en: https://medium.com/

[9] "Página web de wordpress.org." [En línea]. Disponible en: https://wordpress.org/

[10] N. R. Brisaboa, A. Cortiñas, M. R. Luaces, and Óscar Pedreira, "Aplicando scaffolding en el desarrollo de líneas de producto software," 2016.

[11] A. Cortiñas, "Repositorio de github de spl-js-engine." [En línea]. Disponible en: https://github.com/AlexCortinas/spl-js-engine

[12] A. C. Álvarez, *Software product line for web-based geographic information systems*, 2017.

[13] "Guía de vue.js." [En línea]. Disponible en: https://vuejs.org/v2/guide/

[14] "Página web de node.js." [En línea]. Disponible en: https://nodejs.org/es/about/

[15] "Página web de bootstrap." [En línea]. Disponible en: https://getbootstrap.com/docs/4.3/getting-started/introduction/

[16] "Página web de spring." [En línea]. Disponible en: https://spring.io/

[17] "Página web de hibernate." [En línea]. Disponible en: https://hibernate.org/orm/

[18] "Página web de postgresql." [En línea]. Disponible en: https://www.postgresql.org

[19] C. Larman, *Agile and Iterative Development: A Manager's Guide*, 13th ed. Addison Wesley, 2004.

[20] K. Schwaber and J. Sutherland, "The scrum guide™," 2017. [En línea]. Disponible en: https://www.scrumguides.org/scrum-guide.html

[21] "Gitlab agile delivery." [En línea]. Disponible en: https://about.gitlab.com/solutions/agile-delivery/

[22] "Página web de eclipse." [En línea]. Disponible en: https://www.eclipse.org/

[23] "Página web de visual studio code." [En línea]. Disponible en: https://code.visualstudio.com/

[24] "Página web de maven." [En línea]. Disponible en: https://maven.apache.org/

[25] "Página web de junit." [En línea]. Disponible en: https://junit.org/junit4/

[26] "Página web de jacoco." [En línea]. Disponible en: https://www.eclemma.org/jacoco/

[27] "Página web de npm." [En línea]. Disponible en: https://www.npmjs.com/about

[28] "Página web de dbeaver." [En línea]. Disponible en: https://dbeaver.io/

[29] "Página web de balsamiq." [En línea]. Disponible en: https://balsamiq.com/wireframes/

[30] "Página web de magicdraw." [En línea]. Disponible en: https://www.nomagic.com/products/magicdraw

[31] "Página web de draw.io." [En línea]. Disponible en: https://www.draw.io/

[32] "Página web de microsoft project." [En línea]. Disponible en: https://products.office.com/es-es/project/project-and-portfolio-management-software

[33] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[34] F. J. van der Linden, K. Schmid, and E. Rommes, *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer, 2007.

[35] K. Pohl, G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer, 2005.

[36] I. Sommerville, *Ingeniería de Software*, 9th ed. Addison Wesley, 2011.

[37] "Spring security." [En línea]. Disponible en: https://spring.io/projects/spring-security

[38] "Guía de vue router." [En línea]. Disponible en: https://router.vuejs.org/

[39] "Guía de vuex." [En línea]. Disponible en: https://vuex.vuejs.org/

[40] "Repositorio de github de axios." [En línea]. Disponible en: https://github.com/axios/axios

[41] "Documentación de spring framework." [En línea]. Disponible en: https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html