Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN INGENIERÍA DE SOFTWARE

Aplicación web sobre sitios de interés cultural y turístico de Galicia

Estudiante: Junior Paradelo Álvarez
Dirección: Óscar Fresnedo Arias
José Pablo González Coma

A Coruña, febrero de 2021.

A la vida, por demostrarme una vez más que es la dosis lo que hace al veneno.

Agradecimientos

A mis padres, por estar siempre ahí y proporcionarme todas las oportunidades que he necesitado en mi vida. Sobretodo a ti, mamá, porque con esto lo conseguimos.
A mi hermana, por todas las cosas que me ha enseñado y por estar siempre ahí cuando es necesario.
A ambos directores, por entender la situación y ayudar desde el principio, haciendo que este proyecto haya sido más sencillo y llevadero a lo largo del tiempo.
Y a la gente que está… porque aunque parezca poco, es mucho.

Resumen

El objetivo de este trabajo de fin de grado ha sido el de desarrollar una aplicación que ofrezca una gran variedad de información sobre los sitios culturales naturales e históricos más importantes de Galicia, y la posibilidad de una interacción entre usuarios para fomentar su continuo crecimiento y desarrollo.
Para alcanzar este objetivo ha sido necesario realizar un análisis sobre la importancia del patrimonio histórico y cultural de Galicia. En ese análisis, se ha descubierto que se podría fomentar en gran parte el turismo con una aplicación que mostrase las características, detalles y localizaciones de los lugares con un importante rasgo histórico-cultural.
En el desarrollo de esta aplicación web se han utilizado las tecnologías de código abierto como Java, Spring e Hibernate para la creación de un servidor donde desarrollar la parte lógica de la aplicación. Este servidor cuenta con un mapeo a un sistema gestor de base de datos (SGBD) relacional creado con PostgreSQL para el almacenamiento de la información. Además de esto, implementa un servicio REST para la comunicación con un cliente web basado en Vue.js el cual ofrecerá una interfaz amigable con la que interactuar para llevar a cabo las funcionalidades disponibles.
Este trabajo de fin de grado se ha gestionado siguiendo una metodología iterativa e incremental para el desarrollo de software que permite acortar significativamente los ciclos de desarrollo y asegurando que al final de cada ciclo se dispondrá de un software funcional.

Abstract

The objective of this final degree project is to develop an application that manages the location of the most important cultural and historical sites in Galicia, as well as the interaction between users for their continuous growth and development.
To achieve this objective it was necessary to carry out an analysis on the importance of the historical and cultural heritage of Galicia. It was discovered that tourism could be largely promoted with an application that promotes and shows the characteristics, details and locations of places with an important historical-cultural trait.
In the development of this web application, open source technologies such as Java, Spring and Hibernate have been used to create a server where to develop the logical part of the application. This server has a mapping to a relational database management system (DBMS) created with PostgreSQL for information storage. In addition to this, it implements a REST service for communication with a web client based on Vue.js which will offer a friendly interface with which to interact to carry out the available functionalities.

This final degree project has been managed following an iterative and incremental methodology for software development that allows to significantly shorten development cycles and ensure that functional software will be available at the end of each cycle.

Palabras clave:

- Java
- Spring
- Hibernate
- Servicio REST
- Web
- Código abierto
- Javascript
- PostgreSQL
- Vue.js
- Leaflet
- TailwindCSS
- Sitio de interés cultural y turístico
- Mapa
- Scrum

Keywords:

- Java
- Spring
- Hibernate
- REST service
- Web
- Open source
- Javascript
- PostgreSQL
- Vue.js
- Leaflet
- TailwindCSS
- Site of cultural and tourist interest
- Map
- Scrum

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introdución

1.1 Motivación

Teniendo en cuenta los estudios realizados por Exceltur [2] en el año 2017, el turismo ha generado 6.341 millones de euros en Galicia, el 10,4% del conjunto de su economía, y ha dotado de 109.050 empleos (11% del empleo total de la comunidad).
Esto ha ido creciendo y según los datos oficiales de la oficina de turismo de Galicia [3], esta comunidad registraba en el año 2019 los mejores datos turísticos de su historia con casi 11 millones de pernoctaciones y cerca de 5,1 millones de turistas. Esto supuso que Galicia fuese la comunidad autónoma española donde más creció el turismo en 2019. En concreto, un 6%, muy por encima del 1% de crecimiento estatal. Estos datos la sitúan en una posición cada vez más competitiva para el sector turístico.
Considerando la importancia del turismo y el cambio en los gustos de la población, que ahora muestra una mayor preferencia por las zonas rurales respecto a las urbanas, se ha desarrollado esta aplicación web para el impulso del turismo rural e histórico gallego, con la intención de mejorar la búsqueda de entornos de turismo de naturaleza, bienes históricos, y culturales adaptados a los gustos de cada persona.
Para ello, resulta fundamental incorporar la información precisa y completa sobre cada sitio, habilitar filtros que ayuden al usuario, y ofrecer la posibilidad de crear una comunidad activa de usuarios que puedan opinar sobre cada lugar, comentando, puntuando y guardando en diversas listas todos aquellos sitios que le hayan gustado o que tengan pendiente de visitar en un futuro próximo.

Hoy en día, la forma más utilizada para acceder a la información correspondiente a los sitios culturales es mediante la navegación web. Aunque existen numerosas páginas con información diversa sobre lugares de interés que se pueden visitar, la mayoría de las veces no integran toda la información necesaria o útil para ayudar al usuario en su decisión. Muchos de

estos recursos tienen un enfoque diferente, haciendo que el tipo de información que presentan sobre un mismo lugar sea diferente, lo que muchas veces obliga al usuario a visitar varias web para obtener una visión global del lugar sobre en que desea informarse. En general, estas webs tampoco dan al usuario la posibilidad de almacenar información personal sobre ellos, bien sea guardando sus sitios favoritos o estableciendo una lista de sitios pendientes para un futuro.
Por todo esto, el objetivo principal del trabajo ha sido desarrollar una aplicación web que permita la elección de los sitios perfectos para las necesidades de cada usuario concreto, conocer la máxima información de cada uno de ellos sin tener que visitar varias webs, junto con la creación de una comunidad para la mejora y descubrimiento de todas estas zonas (opinando y puntuándolas).
La aplicación contará además con un mapa interactivo en el que el usuario con un simple vistazo podrá ver una determinada ubicación, o sitio de interés, que le llame la atención descubrir y desplegar una amplia información de todo tipo del mismo.

1.2 Objetivos

Para poder alcanzar el objetivo principal de este trabajo de fin de grado descrito en la sección anterior, se han planteado diversos objetivos específicos:

- Es interesante que la aplicación llegue a cualquier tipo de público, por lo que dispondrá de **funcionalidades para usuarios anónimos**, que estarán limitadas a visualizar el contenido publicado, con algunas restricciones para incentivar el registro.
- **Facilitar la búsqueda de sitios con interés cultural e histórico** según la **categoría** que el usuario precise; ya sea un dólmen, un hórreo, un embalse, una iglesia, un mirador, una fervenza, etc.
- Crear una **comunidad de usuarios** que fomenten el turismo y la promoción de las zonas naturales y patrimonio gallego gracias a sus **comentarios, puntuaciones** y experiencias, además de que puedan guardar sus sitios favoritos o poder marcar un sitio para descubrir.
- **Facilitar información detallada y relevante de cada sitio**; ya sea el tipo de vehículo en el que se puede llegar, si hay zona de aparcamiento disponible (en caso de que haya, la ubicación concreta), el nivel de accesibilidad al lugar, el tipo de calzado recomendado, restricciones actualizadas de la zona, etc.
- Disminuir el tiempo de búsqueda de sitios integrando un **mapa interactivo** en el cual el usuario podrá visualizar y descubrir zonas que puedan resultar de su interés.

1.3 Estructura del documento

El presente documento se compone de diez apartados que describen las tecnologías y metodologías empleadas en el proyecto, además de explicar el desarrollo de este.

1.3.1 Apartados

En esta parte se describirá brevemente la estructura y contenido de los diez apartados:

- **Introducción**
  Se analiza la motivación y los objetivos a cumplir en el proyecto. Además de esto, se describe la estructura de la memoria.
- **Estado del arte**
  Se muestran sitios web ya existentes que ayudan al usuario a la hora de hacer turismo. También se analizan sus características principales.
- **Fundamentos tecnológicos**
  Se exponen las tecnologías empleadas para la realización del proyecto.
- **Metodología**
  Se describe la metodología de desarrollo *Scrum*, empleada en la organización del desarrollo del proyecto. Se emplea una versión de Scrum adaptada para este proyecto y se describen los *Sprints* o fases en las que se dividió el desarrollo.
- **Análisis**
  Se realiza un análisis más en profundidad de los requisitos del sistema.
- **Diseño**
  Se incluye el diseño conceptual de la aplicación y de las partes de la misma.
- **Implementación y pruebas**
  Se explica las decisiones y el trabajo relacionado con el software empleado.
- **Solución desarrollada**
  Se muestran las características principales de la aplicación y cómo está creada.
- **Conclusiones y trabajo futuro**
  Se exponen las conclusiones del proyecto, las líneas futuras de desarrollo y las posibles funcionalidades a añadir.
- **Bibliografía**
  Se recogen todas las referencias a la información empleada en el proyecto y memoria.

Capítulo 2

Estado del arte

2.1 Estado del arte

A partir de los objetivos propuestos en la sección anterior, el primer paso que se ha realizado ha sido una búsqueda de herramientas similares ya disponibles, que ofreciesen funcionalidades parecidas a las que se han propuesto en los objetivos, con el fin de ver tanto su funcionamiento como sus puntos débiles y así tener la posibilidad de introducir mejoras de cara a la aplicación a desarrollar. A continuación, se mostrarán una serie de aplicaciones web que son utilizadas en la actualidad por los usuarios para consultar sitios turísticos que visitar.

2.1.1 Galicia máxica

Entre las aplicaciones que se han encontrado, la herramienta más parecida a lo que se buscaba desarrollar con este proyecto es una web llamada ”Galicia máxica” [4] (figura 2.1), la cual muestra una serie de información y ofrece diferentes recursos a los que se puede acceder directamente sin necesidad de registro previo. Es una aplicación principalmente destinada a todas aquellas personas que quieren conocer los sitios más populares que se pueden visitar en Galicia.

Esta aplicación web cuenta con un buscador, como podemos ver en la figura 2.2, para encontrar el sitio que se desee. Este buscador realiza la búsqueda en todos los *posts* de la aplicación web, pero no permite filtrar en ningún momento, algo negativo para los usuarios cuando se trata de realizar una búsqueda más avanzada sobre un determinado sitio.
La web también cuenta con una página donde se muestran los últimos sitios que se han introducido en la aplicación, pero con el mismo inconveniente que se comentaba antes, es decir, el hecho de no poder filtrar por ningún tipo ni categoría. Por lo tanto, al usuario se le muestra mucha cantidad de información que le puede resultar innecesaria y distraerlo en su búsqueda. Además de esto, la aplicación cuenta con un apartado en el que se muestra un mapa de Galicia

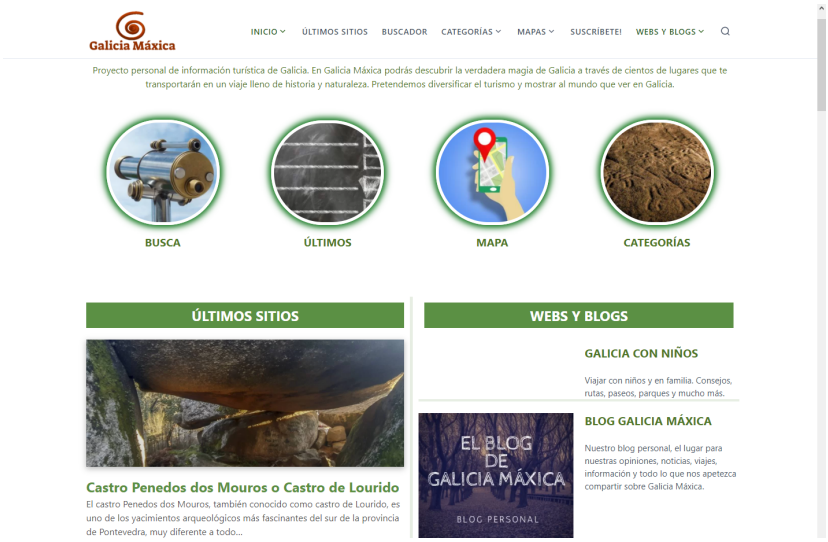

Figura 2.1: Web de Galicia máxica

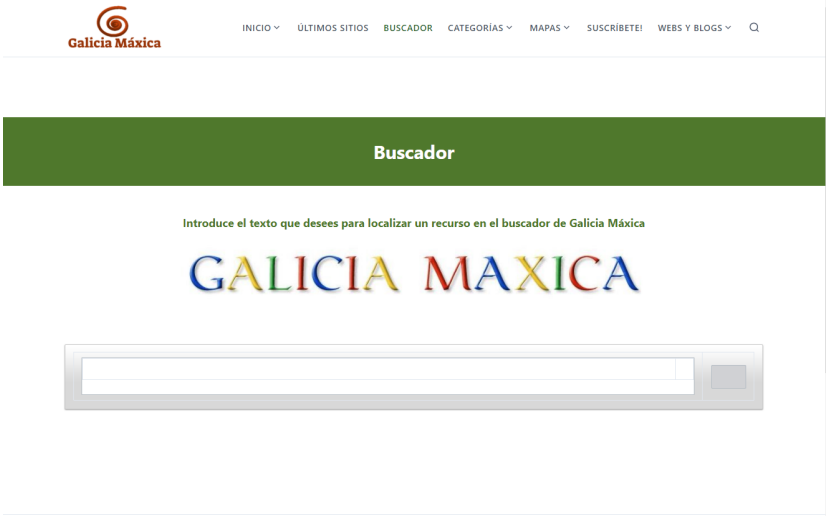

Figura 2.2: Buscador de Galicia máxica

(figura 2.3), en el que se puede ver todo tipo de marcadores que inundan el mapa y no lo hacen todo lo práctico que debería ser.

Todos estos puntos débiles se han tenido muy en cuenta para realizar una aplicación más útil y amigable para el usuario final, con el objetivo de facilitarle la búsqueda de sus sitios de interés y de ayudarlo resaltando aquella información que puede ser más relevante.

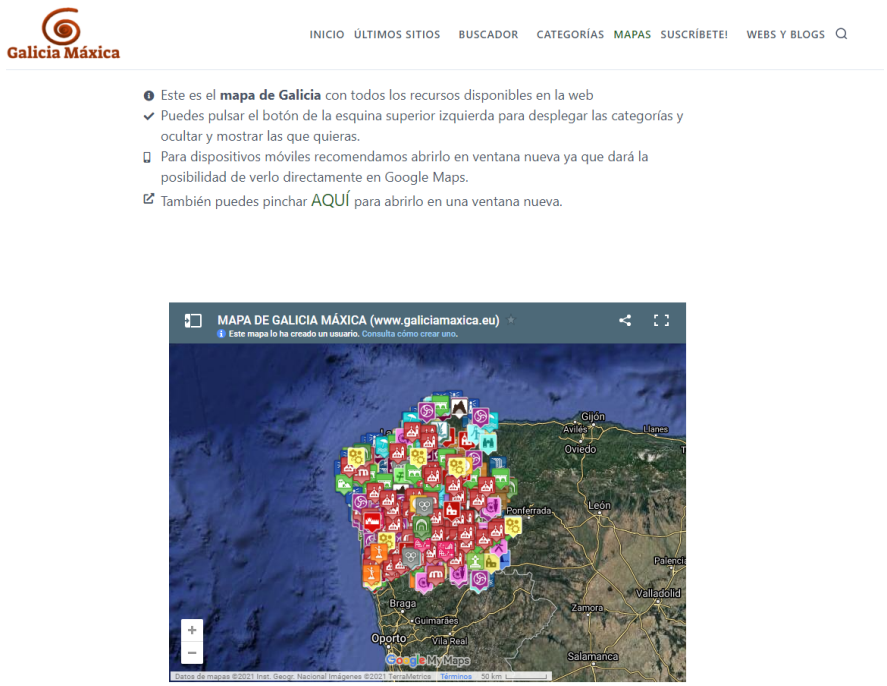

Figura 2.3: Mapa de Galicia máxica

2.1.2 Tripadvisor

En la figura 2.4 se puede ver una captura de Tripadvisor [5] que es la plataforma de viajes más grande del mundo y ayuda a los viajeros a planificar mejor sus futuros viajes. Tripadvisor permite a los usuarios, entre otras cosas, realizar búsquedas de sitios, hoteles, vuelos o excursiones.
Uno de los puntos fuertes de esta aplicación web es que en la información de cada sitio se pueden encontrar reseñas y valoraciones, puestas por otros usuarios, con el fin de tener una comunidad más informada a la hora de realizar un viaje.

Tripadvisor no permite interactuar con ningún tipo de mapa. Esto es un punto negativo ya que, para un turista, resulta de gran utilidad porque puede tener algún sitio en mente al que ir pero que también le gustaría ver de manera interactiva y visual qué sitios pueden estar cerca y planificar un viaje recorriendo varios sitios.

2.1.3 Minube

Minube [6] es una aplicación para compartir los rincones que cualquier persona descubre cuando viaja. Esto permite crear una comunidad de viajeros por todo el mundo con el fin de ayudarse y compartir información entre ellos sobre los sitios que visitan. En la figura 2.5

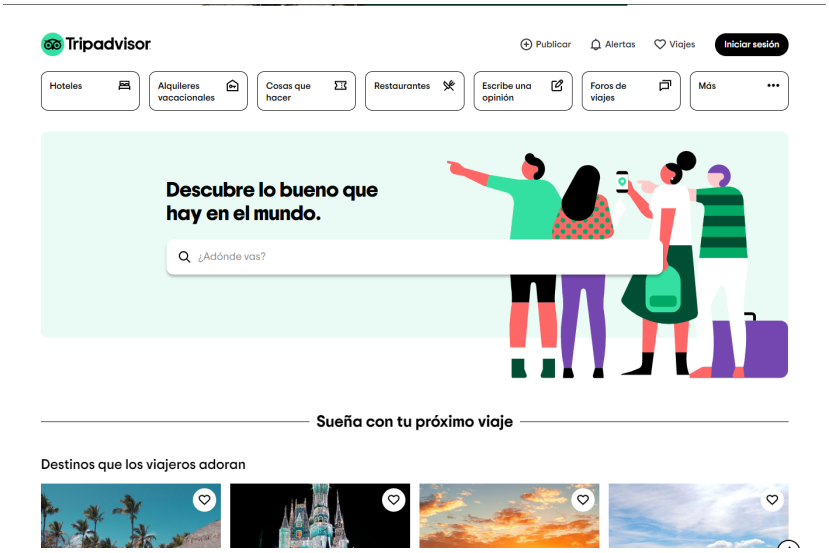

Figura 2.4: Web de Tripadvisor

podemos ver una imagen del portal de la aplicación.

La comunidad de usuarios es lo que le da valor a Minube, ya que todos los sitios que integran están llenos de reseñas de usuarios contando su experiencia para ayudar a otros. Por contra, al basarse en comentarios y experiencias de usuarios tiene como punto negativo la no existencia de información detallada de los sitios. Algunos cuentan únicamente con una ubicación y no existe información sobre cómo se puede llegar ni lo que es necesario para el viaje.

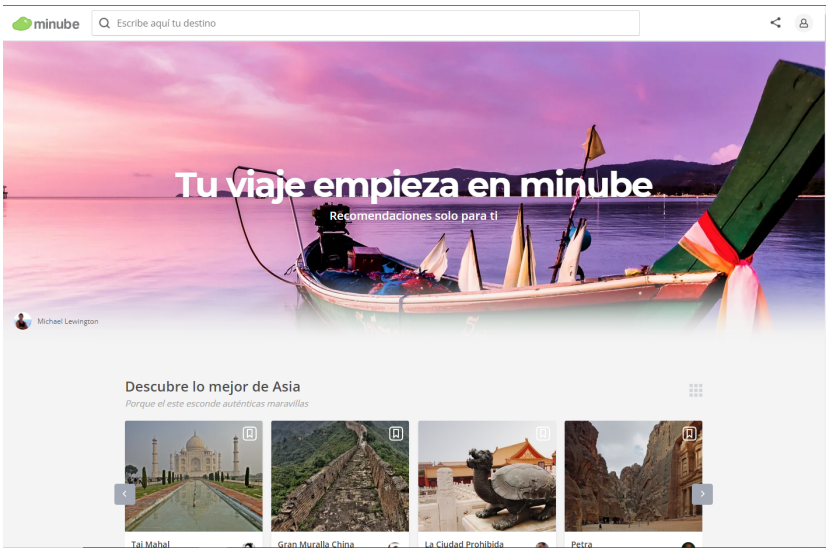

Figura 2.5: Web de Minube

2.1.4 Portal oficial de turismo de España

Con el propósito de aprovechar el gran potencial que tiene el turismo como generador de bienestar social y su impacto en la economía, el portal oficial de turismo de España [7] tiene como objetivo dar a conocer el patrimonio cultural e histórico de este país para atraer a potenciales visitantes y generar valor añadido para el sector turístico.
La principal limitación que tiene esta herramienta es que, a nivel gallego, únicamente podemos encontrar información sobre Santiago de Compostela y no nos ofrece más alternativas. Podemos ver una imagen del portal en la figura 2.6.

Figura 2.6: Web del portal oficial de turismo de España

2.1.5 Portal oficial de turismo de Galicia

El portal oficial de turismo de Galicia [8] (figura 2.7) busca ofrecer un servicio integral para facilitar la gestión de la visita y estadía de un turista en Galicia, tanto para prepararla como durante la misma. Aunque a priori puede parecer interesante y a pesar de que está centrada en la comunidad gallega, una vez navegas por la web no tiene muchos destinos que elegir y no cubre todos los puntos que necesita un turista a la hora de realizar un viaje.

2.1.6 Sitiosturísticos

Aunque en esta aplicación web no está especializada en Galicia, sino en las comunidades más típicas de cada país (figura 2.8), nos da una idea de qué cosas son necesarias e imprescindibles en general para un viajero. En sitiosturísitcos [9] cuando miramos la información de

Figura 2.7: Web del portal oficial de turismo de Galicia

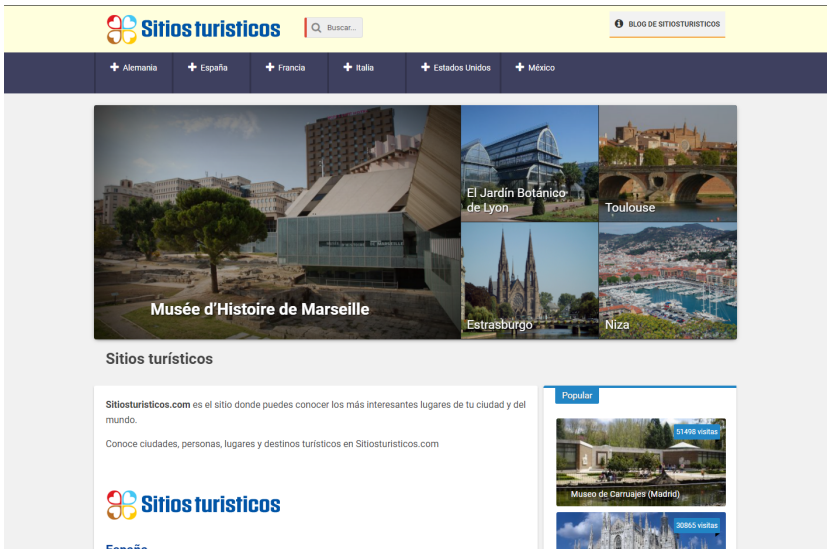

Figura 2.8: Web de sitiosturisticos

las ciudades, nos muestran los sitios más importantes que tienen mostrando una información detallada y bastante completa de cada lugar.

2.2 Conclusiones

En esta sección se van a mostrar las conclusiones que se han obtenido al revisar todas las aplicaciones de la sección anterior 2.1. En ella se mostrará lo que hemos utilizado para

inspirarnos y también las deficiencias que se han detectado y se han tratado de mejorar con el desarrollo de nuestra aplicación.

Puntos favorables:

- Todas cuentan con un buscador, algo muy importante en una web con el fin de ayudar a descubrir zonas o lugares a un usuario.
- La web de ”Galicia Máxica” cuenta con una página en dónde se muestran los últimos sitios introducidos en la aplicación. Esto es algo muy útil para la comunidad que utiliza con frecuencia la aplicación ya que con un simple vistazo puede encontrar algo que le interese.
- Que una aplicación cuente con un mapa interactivo es algo muy atractivo a la hora de utilizar, ya que permite buscar de manera sencilla sitios por una determinada zona.
- Una aplicación que cuente con posibilidad de realizar comentarios o valorar y puntuar sitios crea una comunidad que puede aportar más información de la que se ha detallado del propio sitio.

Deficiencias encontradas:

- En los buscadores no cuentan con muchos filtros que pueden ser muy útiles para el usuario, ya que no tiene que demorar mucho tiempo en ver información que no le resulta útil.
- En ”Galicia Máxica” a pesar de que cuenta con un mapa interactivo, el mostrar todos los puntos de todos los sitios existentes al mismo tiempo crea una nube de puntos que no es fácilmente manejable por los usuarios.
- Las aplicaciones anteriores no cuentan con mucha información detallada, ni de qué posibilidades o restricciones, por ejemplo de accesibilidad, tienen los sitios a la hora de visitarlos.
- Algunos sitios web se centran en dar información de los puntos más importantes de cada comunidad, impidiendo que así prospere el turismo en otras zonas menos importantes que pueden tener encanto para los viajeros.
- No se pueden guardar sitios favoritos, o tener una lista para guardar sitios que interesan al usuario para visitar en un futuro. Esto provoca que si un usuario encuentra un sitio nuevo navegando por la web puede que después de un tiempo no lo recuerde o le cueste mucho encontrarlo otra vez, perdiendo así la oportunidad de visitarlo.

Capítulo 3

Fundamentos tecnológicos

En este capítulo, se van a listar y a explicar las distintas tecnologías que se han utilizado para el desarrollo del proyecto.

3.1 Herramientas

- **PgAdmin [10]:** Cliente de conexión y gestión de bases de datos. PgAdmin es la plataforma de desarrollo y administración de código abierto más popular y rica en funciones para PostgreSQL [11], que es la base de datos de código abierto más avanzada del mundo. Una alternativa que se había contemplado en este caso era la posibilidad de utilizar DBeaver [12], pero al final se optó por pgAdmin por la fácil integración con la base de datos de PostgreSQL.
- **SourceTree [13]:** Cliente de configuración y gestión de repositorios git. Se ha optado por esta aplicación porque simplifica la forma de interactuar con los repositorios git, realizando todas las operaciones necesarias de una manera sencilla a través de un interfaz de usuario en vez de por línea de comandos.
- **Postman [14]:** Herramienta que se utiliza, sobre todo, para el *testing* de la API REST. Esta herramienta nos ofrece un conjunto de utilidades adicionales para poder gestionar las *API* de una forma más sencilla. Entre otras funcionalidades, proporciona herramientas para documentar las APIs o para realizar una monitorización de ellas.

3.2 Entornos de desarrollo

- **Spring tool suite [15]:** Entorno de desarrollo integrado (IDE) para el desarrollo de la parte del *backend* en aplicaciones Java. Debido a que la aplicación que se va a desarrollar en este proyecto está basada en Spring, este entorno nos ofrece todas las herramientas necesarias para elaborar la aplicación de una manera más óptima.

- **Visual Studio Code [16]:** IDE para el desarrollo de aplicaciones web (desarrollo del *frontend*). Visual Studio Code es un editor de código fuente desarrollado por Microsoft gratuito y de código abierto. Entre sus muchas características incluye soporte para la depuración de código en muchos lenguajes de programación; control integrado de git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código, etc. Por tanto, su uso simplifica mucho el trabajo al desarrollador.

3.3 Tecnologías

- **Java [17]:** Es un lenguaje de programación orientado a objetos. La idea de Java es que puedan realizarse programas con la posibilidad de ejecutarse en cualquier contexto, en cualquier ambiente, siendo su portabilidad una de sus principales características. Para este proyecto se ha utilizado AdoptOpenJDK [18], la versión open source de Java (cuya licencia es de Oracle [19]), para realizar una aplicación open source completa.

- **Spring [20]:** Es un *framework* para el desarrollo de aplicaciones de código abierto para la plataforma Java y que se basa en el principio de inversión de control. Spring tiene contribuciones de todos los grandes nombres de la tecnología, incluidos Alibaba, Amazon, Google, Microsoft y más.
  La gran variedad de extensiones y bibliotecas de terceros que existen para Spring permite a los desarrolladores crear casi cualquier aplicación imaginable. El *framework* de Spring hace uso de la inversión de control (IoC) e inyección de dependencia (DI) para facilitar la creación de todo tipo de servicios como, por ejemplo, microservicios seguros, reactivos y basados en la nube para la web o flujos de datos de transmisión complejos para la empresa. Los proyectos de Spring admiten además el modelo de programación reactiva (sin bloqueo) para lograr una mayor eficiencia.
  Otro aspecto interesante es que Spring Boot [21] transforma por completo la forma en que se abordan las tareas de programación de Java, buscando optimizar la experiencia del desarrollador. Spring Boot combina utilidades como un contexto de aplicación o un servidor web integrado y autoconfigurado para hacer que el desarrollo de microservicios sea muy sencillo. Para ir aún más rápido, se puede combinar Spring Boot con el extenso conjunto de bibliotecas, servidores, patrones y plantillas de soporte de Spring Cloud, para implementar de forma segura arquitecturas completas basadas en microservicios en la nube, en un tiempo récord. La productividad de los desarrolladores es el superpoder de Spring.
  En este sentido, Spring Boot ayuda a los desarrolladores a crear aplicaciones con facilidad y con mucho menos esfuerzo que otros paradigmas de la competencia.

- **Hibernate[22]:** Es un *framework* *Object-relational mapping (ORM)* que facilita el mapeo de atributos entre la base de datos y el modelo de datos de la aplicación. Además de su propia API "nativa", Hibernate también tiene una implementación de la especificación de Java persistence API (JPA). Como tal, se puede utilizar fácilmente en cualquier entorno compatible con JPA, incluidas las aplicaciones *Java enterprise edition (Java SE)* o los servidores de aplicaciones *Java standard edition (Java EE)*.
  Hibernate permite desarrollar clases persistentes siguiendo las características propias de la orientación a objetos como herencia, polimorfismo, asociación, composición, etc. Hibernate no requiere interfaces ni clases base para crear las clases persistentes y permite que cualquier clase o estructura de datos pueda ser persistente.
  Además, admite la inicialización diferida, numerosas estrategias de búsqueda y bloqueo optimista con control de versiones automático y sellado de tiempo. Hibernate no requiere tablas o campos de base de datos especiales y genera gran parte del código SQL [23] necesario para acceder en a la base de datos en el momento de la inicialización del sistema en lugar de en tiempo de ejecución. De esta forma, ofrece un rendimiento superior con respecto a usar código de la Java Database Connectivity (JDBC) directamente, tanto en términos de productividad del desarrollador como de rendimiento en tiempo de ejecución.
  Por último, Hibernate fue diseñado para funcionar en un clúster de servidor de aplicaciones, por lo que ofrece una arquitectura altamente escalable, y es altamente configurable y extensible.

- **Hibernate Spatial [24] [25]:** Es una extensión genérica a Hibernate para el manejo de datos geográficos. Los datos geográficos incluyen la representación de entidades como un Punto, Línea o Polígono. Sin embargo, este tipo de datos no son parte de la especificación JDBC, por lo que Java Topology Suite (JTS) [26] se ha convertido en un estándar para la representación de tipos de datos espaciales. Hibernate Spatial también soporta Geolatte-geom, una biblioteca que tiene algunas características interesantes que no están disponibles en el JTS.

- **Vue.js [27]:** Es un *framework* progresivo para construir interfaces de usuario. Su núcleo es bastante pequeño y se escala a través de *plugins*. Se caracteriza por englobar en un mismo archivo el código HTML, CSS y JavaScript de las vistas. Se ha decidido utilizar este *framework* para crear el cliente con el objetivo de que el autor del proyecto adquiera nuevos conocimientos sobre ciertas tecnologías que no ha sido visto en ningún momento durante el grado.
  Se ha optado por utilizarlo en el *frontend* ya que es un *framework* que se basa en componentes que se comunican con el servidor mediante diferentes métodos de manera

asíncrona. En cada componente se separan los dos elementos principales: los datos y los métodos. Esto permite al desarrollador tener todo ubicado de una manera más clara.

- **Tailwind [28]:** Tailwind CSS es un *framework* que permite un desarrollo ágil, basado en clases de utilidad que se pueden aplicar con facilidad en el código HTML y unos flujos de desarrollo que permiten optimizar mucho el peso del código CSS. Se ha decidido utilizar este *framework* CSS para crear una aplicación web visualmente atractiva sin las limitaciones de usar el *framework* de Bootstrap [29], donde los recursos visuales son bastante limitados y restrictivos.

- **Leaflet [30] [31]:** Librería JavaScript para ver y crear mapas. Leaflet es la biblioteca JavaScript de código abierto más potente para la creación de mapas interactivos compatibles con dispositivos móviles. Tiene todas las funciones de mapeo que la mayoría de los desarrolladores necesitan. Además, está diseñada teniendo en cuenta la simplicidad, el rendimiento y la facilidad de uso. Funciona de manera eficiente en las principales plataformas móviles y de escritorio, se puede ampliar con muchos complementos, tiene una API fácil de usar y bien documentada, y un código fuente simple y legible.

- **JSON web token (JWT) [32]:** Estándar para transmitir información de usuario de forma segura entre un cliente y un servidor. Un JSON Web Token es un *token* de acceso estandarizado que permite el intercambio seguro de datos entre dos partes. Contiene toda la información importante sobre una entidad, lo que implica que no hace falta consultar una base de datos ni que la sesión tenga que guardarse en el servidor (sesión sin estado).
  Por este motivo, los JWT son especialmente populares en los procesos de autenticación. Con este estándar es posible cifrar mensajes cortos, dotarlos de información sobre el remitente y demostrar si este cuenta con los derechos de acceso requeridos. Los propios usuarios solo entran en contacto con el *token* de manera indirecta: por ejemplo, al introducir el nombre de usuario y la contraseña en una interfaz.
  Se ha decidido utilizar JWT para facilitar las siguientes operaciones en nuestra aplicación:

  - Autorización: este es el escenario más común para usar JWT. Una vez que el usuario haya iniciado sesión correctamente, cada solicitud posterior incluirá el JWT correspondiente, lo que permitirá al usuario acceder a rutas, servicios y recursos que están permitidos con ese *token*. El inicio de sesión único es una función que utiliza ampliamente JWT en la actualidad, debido a su pequeña sobrecarga y la posibilidad de usarse fácilmente en diferentes dominios.
  - Intercambio de información: el uso de estos *tokens* es una buena forma de trans-

mitir información de forma segura entre las partes implicadas en la comunicación. Debido a que los JWT se pueden firmar, por ejemplo, utilizando pares de claves públicas / privadas, se puede garantizar que los remitentes son quienes dicen ser. Además, como la firma se calcula utilizando el encabezado y la carga útil, también se puede verificar que el contenido no haya sido manipulado.

- **Axios [33] [34]:** Cliente HTTP basado en promesas de javascript [35] para enviar peticiones de forma asíncrona a un servicio REST. Axios está pensado para facilitar el consumo de servicios web que devuelvan datos JSON.
  Para trabajar con Axios usamos el API de promesas [36]. Esto quiere decir que, cuando se reciba la respuesta del servidor se ejecutará una función *callback* configurada en *"then"* y cuando se produzca un error (por ejemplo una respuesta de recurso no encontrado) se ejecutará la función *callback* definida por el *"catch"*.

- **CSS:** Es un lenguaje de estilo para describir el diseño de una página web (colores, disposición de elementos y fuentes). Aunque es independiente de HTML, suelen emplearse conjuntamente.

- **HTML:** Lenguaje estándar para la creación de páginas web. Los elementos HTML son los bloques de construcción de las páginas HTML y se definen mediante el uso de etiquetas delimitadas por los símbolos <> (*tags*).

- **Javascript [37] [38]:** Es un lenguaje de programación interpretado para la creación de vistas dinámicas, derivado del estándar ECMAScript. Desde el 2012, todos los navegadores modernos soportan completamente ECMAScript 5.1, una versión de JavaScript y pueden interpretar código JavaScript. Este lenguaje se diseñó con una sintaxis similar a C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo, Java y JavaScript tienen semánticas y propósitos diferentes.
  Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en el lado del cliente (*client-side*), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, aunque existe una forma de JavaScript del lado del servidor (*Server-Side JavaScript* o SSJS). Su uso en aplicaciones externas a la web como, por ejemplo, en documentos PDF o aplicaciones de escritorio (mayoritariamente *widgets*), es también significativo.
  Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página con el lenguaje JavaScript, se le proporciona una implementación del *Document Object Model (DOM)* correspondiente a la página que puede ser actualizado dinámicamente con JavaScript. Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de

la aplicación cliente, sin acceso a funciones del servidor. Actualmente es ampliamente utilizado para enviar y recibir información del servidor junto con ayuda de otras tecnologías como *AJAX* [39]. JavaScript se interpreta en el navegador del usuario al mismo tiempo que las sentencias van descargándose junto con el código HTML.

- **Swagger UI [40]:** Esta herramienta permite visualizar e interactuar con los recursos de una API sin necesidad de implementar ninguna lógica adicional. Se genera automáticamente la documentación visual que facilita la implementación del *backend* y el consumo de los servicios del lado del cliente. Utilizando Swagger UI para exponer la documentación de nuestra API, se ha podido ahorrar mucho tiempo lo que nos permite en una metodología ágil cumplir antes con los objetivos.

3.4 Base de datos en un sistema de información geográfica (SIG)

Una base de datos (BD) es el producto de la necesidad humana de almacenar información. Cada base de datos está compuesta de una o más tablas que guardan un conjunto de datos. Cada tabla tiene una o más columnas y filas. Las columnas guardan una parte de la información sobre cada elemento que queramos guardar en la tabla, mientras que cada fila de la tabla se corresponde con un registro. Las bases de datos se pueden clasificar en:

- **Bases de datos relacionales [41]:** La base de datos relacional (BDR) es un tipo de base de datos que cumple con el modelo relacional. En el modelo relacional, la base de datos tiene una estructura rígida, que requiere un tipo concreto para cada valor y exige la inclusión de ciertos campos para realizar una inserción.

- **Bases de datos no relacionales [42] (NoSQL):** El término NoSQL hace referencia a un nuevo enfoque en los Sistemas de Gestión de Bases de Datos (SGBD), donde se permiten la inclusión de datos en un formato flexible que queda a elección del cliente en el momento de la inserción. En este sentido, no se restringe explícitamente la aceptación de las nuevas entradas según los campos que realmente se hayan definido. Ni siquiera es necesario definir con antelación la estructura de los datos o qué campos existen. De este modo, se podría considerar que cada inserción es un objeto independiente, que puede tener o no campos en común con el resto.

Todo el manejo y acceso a los datos almacenados en una BD se realiza mediante los llamados sistemas de gestión de bases de datos (SGBD), que permiten un almacenamiento ordenado y una rápida recuperación de la información. El contenido de una base de datos engloba la información almacenada en archivos a una organización, de manera que los datos se encuentren disponibles.

El SGBD es la parte más importante del software de un sistema de BD. Es una colección de rutinas de software interrelacionadas, cada una con una tarea específica. El objetivo de un SGBD es proporcionar una serie de herramientas para extraer, almacenar y manipular la información de la BD, garantizando además la seguridad de la misma. Todas las peticiones de acceso a la BD se hacen de forma centralizada por medio del sistema, por lo que funciona como interfaz entre los usuarios y la base de datos. Las principales funciones de un SGBD son:

- Definición o descripción. Permite al administrador definir la estructura de los datos, sus interrelaciones y su semántica.
- Manipulación. Permite a los usuarios manejar la base de datos mediante operaciones de consulta y actualizado (inserción, modificación y borrado).
- Utilización. Integra las interfaces de los diferentes tipos de usuario y proporciona al administrador un conjunto de procedimientos y herramientas para la explotación de la base de datos.
- Herramientas y utilidades. Carga y traducción de ficheros, generación de informes, formularios y copias de seguridad, etc.
- Debe garantizar también las funciones de la base de datos, es decir, integridad, confidencialidad, concurrencia y transaccionalidad.

Existen dos formas de representar la información: de forma espacial o de forma nominal. Una **base de datos espacial** [43] es una colección de datos espacialmente referenciados, que actúan como modelo de la realidad. Por lo tanto, son básicamente sistemas donde se guarda información de tipo espacial (coordenadas de lugares, rutas, etc).
De esta forma, un dato espacial será una variable asociada a una localización en el espacio. Normalmente se utilizan datos vectoriales para representarlos que pueden ser expresados mediante tres tipos de objetos espaciales:

- **Puntos:** Es el tipo de objeto espacial más simple, (y el que utilizaremos en nuestra aplicación). Vienen determinados por las coordenadas de un punto definidas en términos de la latitud y la longitud.
- **Líneas:** Son objetos espaciales abiertos que cubren una distancia como, por ejemplo, redes de transporte o vías de comunicación.
- **Polígonos:** Se corresponden con objetos espaciales que representan áreas geográficas (países, regiones, lagos, etc).

Teniendo en cuenta todo lo anterior, en el momento de elegir una tecnología u otra se ha decidido optar por usar una base de datos relacional SQL debido a que, en el proyecto, existe una estructura bien definida y unos tipos de datos bien definidos. Además de eso, era necesario utilizar una base de datos que permitiera almacenar toda la información geográfica [44] de la aplicación. Por ello, se ha decidido utilizar PostgreSQL, con la extensión de PostGIS, ya que, de forma estandarizada, proporciona una serie de funciones y utilidades para manejar los diferentes tipos de datos en la aplicación.

- **PostgreSQL [11]:** Es un servidor de base de datos relacional libre que soporta características propias de la orientación a objetos como puede ser la herencia, tipos de datos, funciones, restricciones, y otras funcionalidades avanzadas relacionadas con las BD como disparadores, definición de reglas e integridad transaccional. Además, se distribuye bajo la licencia BSD, es decir, que el autor permite su modificación y distribución, pero se reserva el derecho de atribución sobre posibles modificaciones. Soporta, a parte de los típicos datos de una BD, grandes ficheros binarios que incluyen archivos de imágenes, sonidos o vídeo.

- **PostGIS [45]:** Es una extensión de bases de datos espaciales para PostgreSQL. Añade soporte para objetos geográficos permitiendo el almacenamiento y manipulación de datos geográficos, además de que las consultas de localización se ejecuten en SQL. PostGIS implementa metadatos y funciones geométricas para el tratamiento de los datos espaciales basándose en el estándar *Open Geospatial Consortium (OGP)*.

3.5 Herramientas y utilidades secundarias

Además de las herramientas principales, se han utilizado varias herramientas para facilitar las distintas tareas relacionadas con el proyecto. Entre ellas, **Opera [46]** que es un navegador web utilizado para ejecutar las diferentes vistas del proyecto y para probar la aplicación a nivel de usuario. **Moqups [47]**, que es una herramienta web que permite diseñar de forma sencilla maquetas o bocetos de interfaces web. En las etapas tempranas del desarrollo de la aplicación se han creado *mockups* para comprender de manera visual cómo se quería diseñar la vista de la aplicación. Para crear los diversos diagramas, se ha utilizado **Draw.io [48]**.
Durante el transcurso del desarrollo de la aplicación, se ha utilizado **Notion [49]**, que es un sistema de organización de la información para maximizar la productividad. Notion ha sido utilizado por el autor de este proyecto para llevar un control sobre las tareas pendientes a desarrollar en cada *sprint*, y también para guardar notas que fuesen relevantes sobre el proyecto durante las reuniones al final de cada *sprint*.

Capítulo 4

Metodología y planificación

En este capítulo se va a exponer la metodología que se ha utilizado para el desarrollo del proyecto y se detallarán las iteraciones o fases en las que se ha dividido. También se incluye una estimación de costes sobre el proyecto.

4.1 Metodología de desarrollo

Para la realización de este proyecto se ha escogido una metodología ágil para el desarrollo de software como es **Scrum [50] [51]**. Esta metodología incremental se caracteriza por dividir el ciclo de vida del proyecto en iteraciones que se suceden en el tiempo y que se denominan *sprints*. Cada una de estas iteraciones supone un incremento en la funcionalidad hasta alcanzar un producto entregable.
En este tipo de metodología se le permite al desarrollador sacar ventaja de lo aprendido de la iteración anterior y ofrece una rápida adaptación a los cambios. De esta forma, se pueden producir de forma rápida y continua incrementos del producto para poder ofrecérselos al cliente y así obtener *feedback* del mismo para detectar errores rápidamente. En la figura 4.1 se pueden apreciar gráficamente los principios de desarrollo iterativo en los que se basa Scrum.
Scrum es una metodología adecuada para este proyecto debido a que se va a trabajar con unos requisitos claros desde el principio, y Scrum nos permite obtener productos funcionales desde las etapas más tempranas del desarrollo de la aplicación. Esto nos da la ventaja de crear las funcionalidades de mayor beneficio al principio y después ir añadiendo funcionalidades secundarias de una manera progresiva durante los diferentes sprints.

Las iteraciones comienzan planificando el trabajo a realizar teniendo en cuenta la experiencia en las iteraciones previas, ya que de este modo será más sencillo estimar el tiempo necesario. Al final de cada una de ellas, se realiza una revisión del incremento del producto obtenido y se toman las decisiones necesarias en caso de detectar cualquier problema.

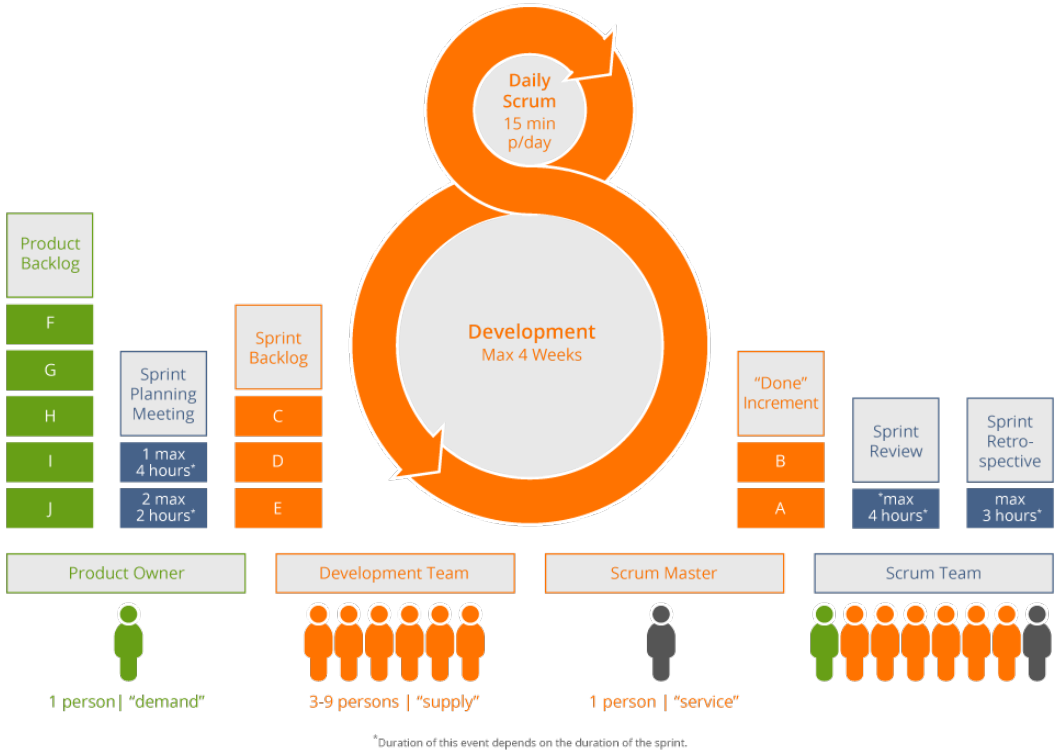

Figura 4.1: Desarrollo Scrum [1]

Para la puesta en práctica de esta metodología, se ha decidido adaptar Scrum a las características específicas del proyecto, teniendo en cuenta que este marco de trabajo está concebido para equipos de desarrollo formados por entre cinco y nueve personas, y en este trabajo participan solamente tres.
Una vez que hemos definido el tipo de metodología que se va a utilizar, vamos a explicar los diferentes elementos que componen Scrum y su adaptación a las necesidades del proyecto:

4.1.1 Equipo

Los equipos de *Scrum* son auto-organizados y multifuncionales. Los equipos auto-organizados eligen la forma en la que quieren trabajar, en lugar de ser dirigidos por personas ajenas al equipo. Esto implica que todos tienen las capacidades y competencias necesarias para realizar el trabajo sin depender de otros. Se distinguen tres **roles** claros:

- **Dueño del producto (*Product Owner*):** Es responsable de maximizar el valor del producto resultante del trabajo del equipo de desarrollo. La forma en que esto se hace puede variar ampliamente entre las organizaciones, los equipos de Scrum y los individuos. El *Product Owner* puede hacer el trabajo anterior o hacer que el equipo de desarrollo lo haga. El *Product Owner* debe identificar los requisitos del proyecto para definir el *Product*

Backlog, se asegura que estén claros para todo el equipo de desarrollo, y busca optimizar el valor resultante del trabajo.

- **Equipo de desarrollo (*Development Team*):** Constituido por todos los trabajadores encargados de desarrollar el producto y, por tanto, responsables del resultado final. Conocedores del proyecto Scrum, el equipo ha de ser multidisciplinar, cubriendo todos los campos necesarios para realizar el proyecto. Se debe gestionar, planificar y organizar para alcanzar los objetivos de cada *sprint*. El tamaño recomendado es de entre 3 y 9 individuos, con habilidad para realizar los trabajos requeridos, tanto de análisis, como de diseño, implementación, pruebas, documentación, etc.

- **Scrum Master:** El *Scrum Master* es un una especie de líder del equipo de Scrum pero cuya función no es realmente organizar al resto de trabajadores del proyecto. Es el encargado de eliminar los obstáculos que dificulten la llegada a las metas definidas y verificar que se siguen los principios establecidos por el marco de trabajo.

4.1.2 Eventos de Scrum

Todos los eventos en Scrum deben tener un tiempo determinado, de modo que cada evento tiene una duración máxima. Una vez que comienza un *sprint*, su duración es fija y no se puede acortar o alargar. Los eventos restantes pueden finalizar siempre que se logre el propósito del evento, asegurando que se pasa una cantidad de tiempo adecuada sin permitir pérdidas de tiempo en el proceso.
Los principales eventos que se definen en Scrum son:

- ***Sprint:*** Bloque de tiempo, de tres semanas en el caso de este proyecto, durante el cual se crea un incremento del producto.

- **Revisión del *sprint (Spring Review)* y reunión de planificación del *sprint (Sprint Planning Meeting):*** En la reunión de planificación es necesario definir las funcionalidades del producto que se van a desarrollar en ese *sprint*, mientras que en la revisión del *sprint* se verifica que el incremento obtenido cumple con las expectativas iniciales y se hace una reflexión para identificar posibles fuentes de problemas que pueden retrasar el desarrollo del software. Durante la realización del proyecto, se han adaptado estos dos eventos de forma que ambos se realizaban en una misma reunión para ajustarse al horario de trabajo del autor y reducir así el número de reuniones por motivos de disponibilidad. Durante cada una de estas reuniones, primero se revisaba el trabajo realizado y se hacía una pequeña ejecución de las funcionalidades desarrolladas, además de sugerir mejoras en caso de que fuera necesario. Una vez acabada la revisión del *sprint*, se pasaba a la planificación del siguiente, se formulaban dudas que iban surgiendo y

se proponían posibles cambios, ya que el proyecto fue evolucionando. Estas reuniones tenían una duración de dos horas aproximadamente.

- **Scrum Diario *(Daily Scrum):*** Son reuniones diarias cortas (sobre 15 minutos de duración) para informar del progreso realizado el día anterior y de los próximos objetivos. En estas reuniones son los roles los que tiene la palabra, se define lo hecho el día anterior a la reunión, existencia de obstáculos, posibles soluciones y lo que se espera alcanzar el día siguiente.
  Al adaptarlo a las necesidades del proyecto y al tratarse de una sola persona en el equipo de desarrollo, no se han llevado a cabo de una forma explícita.

- **Retrospectiva del *sprint (Sprint Retrospective):*** Tiene lugar después de la revisión del *sprint* y es convocada por el *Scrum Master*. Todos los integrantes del equipo dan su impresión sobre el transcurso de cada iteración con la finalidad de identificar los aspectos que han ido bien y aquellos que se pueden mejorar de cara a la próxima iteración. En ese sentido, es una buena oportunidad para que el equipo haga una auto-reflexión y cree un plan de mejoras para la siguiente iteración.
  En el caso de este proyecto, las reuniones de planificación y revisión del *sprint* se han llevado a cabo al mismo tiempo, al final de cada *sprint*. No se ha considerado necesario realizar el Scrum Diario y la Retrospectiva ya que el equipo de desarrollo consta de un solo miembro. En cualquier caso, se han aprovechado las reuniones entre el alumno y sus directores para hacer una reflexión consensuada sobre el desarrollo del proyecto con el objetivo de identificar posibles problemas que fueran surgiendo.

4.1.3 Artefactos

Son los elementos que forman parte del ciclo de vida de Scrum. Representan trabajo o valor en diversas formas que son útiles para proporcionar la información clave a todos los participantes del grupo de trabajo.

- **Pila de producto *(Product Backlog):*** Es una lista ordenada pública, que está disponible para todos los involucrados en el proyecto, y que contiene todo lo que se sabe que se necesita incluir en el producto. De esta forma, el *Product Backlog* constituye una fuente única con los requisitos del producto que se va a desarrollar y cualquiera cambio que se quiera introducir debe ser reflejado en ella. En este sentido, el *Product Owner* es responsable del *Product Backlog* y, por tanto, es el único que lo puede editar. Evoluciona a medida que el producto y el entorno en el que se utilizará también evolucionan. Es, por tanto, dinámico, cambia constantemente para identificar lo que el producto necesita para ser apropiado, competitivo y útil. La lista correspondiente al *Product Backlog* debe estar ordenada por prioridades.

- **Pila de pendientes del *sprint (Sprint Backlog):*** Es el conjunto de elementos del *Product Backlog* seleccionados para ser abordados en un determinado *sprint*, además de un plan para entregar el incremento correspondiente y así cumplir con el objetivo del *sprint*. En Scrum, las tareas no son asignadas a ninguna persona o grupo en concreto, sino que los miembros del equipo de trabajo las van seleccionando cuando lo ven adecuado y tienen disponibilidad.

- **Incremento:** Es la suma de todos los elementos del *Product Backlog* completados durante un *sprint* y de los incrementos previos. Por tanto, corresponde con una versión funcional del producto al final de un *sprint* que se irá completando en las siguientes iteraciones. Al final del *sprint*, el nuevo incremento debe aparecer marcado como ”hecho”. En este proyecto, para que fuese catalogado como ”hecho”, la/s funcionalidad/es correspondientes a ese *sprint* tuvieron que ser implementada/s y probada/s.

4.2 Planificación y seguimiento

En esta sección se describirá la planificación inicial del proyecto, así como una estimación de los recursos necesarios (técnicos y humanos) con sus correspondientes costes.

4.2.1 Planificación inicial

Antes de comenzar con el desarrollo propiamente de la aplicación se realizó un trabajo de análisis para sentar las bases del proyecto y para tener una guía que seguir en el momento del desarrollo de la aplicación.
Para ello se realizó un **análisis preliminar** del alcance del proyecto y de las tecnologías a utilizar. Una vez se finalizó ese análisis previo, se volvió a hacer una reunión con los directores para compartir la información y planificar los siguientes pasos: realizar una descripción de los objetivos principales del proyecto, con sus correspondientes requisitos y funcionalidades asociadas. Una vez completada esta parte, se volvió a tener una reunión para concretar la forma de dividir las funcionalidades de la aplicación.
En concreto, se han seguido los siguientes pasos:

- En primer lugar, se ha diseñado un **modelo de datos** para comprender los tipos de datos que debería contener la base de datos y la relación entre ellos.

- Una vez diseñado el modelo de datos, se creó una **pila inicial del producto** con una serie de historias de usuario que representan los casos de uso identificados con el objetivo de tener una referencia clara para escoger las funcionalidades en cada *sprint.* Esta pila de producto ha ido variando a lo largo del proyecto con los cambios que se han considerado necesarios en las diferentes reuniones al final de los *sprints.*

- Teniendo en cuenta las historias de usuario, se consideró necesario elaborar unos **prototipos iniciales de la interfaz de usuario** en conjunto con los casos de uso para tener una idea inicial de las pantallas necesarias para cubrir todas las historias de la pila de producto.

- Por último, se realizó un **estudio de las tecnologías necesarias** para el desarrollo del proyecto para conocer las principales características de cada una y decidir qué alternativas se ajustan a los requerimientos del trabajo.

Para terminar esta planificación inicial, se realizó un diagrama de Gantt básico (figura 4.2), donde se puede ver la distribución de los *sprints* en el tiempo para ajustarse a las fechas de entrega del proyecto. Como se puede apreciar en la figura 4.2, se decidió dividir el desarrollo de la aplicación en 5 *sprints* con una duración de tres semanas cada uno de ellos. Teniendo en cuenta esta planificación inicial, el desarrollo del proyecto empezaría el 02 de Noviembre y terminaría el 08 de Febrero.

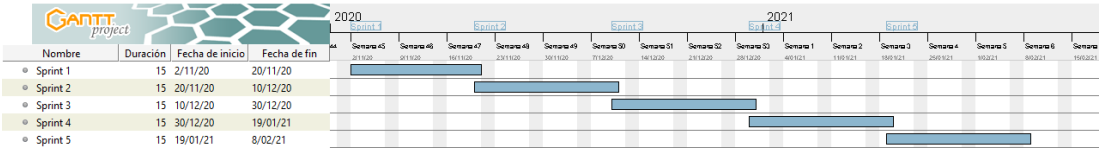

Figura 4.2: Diagrama de Gantt con los sprints

4.2.2 Seguimiento

Una vez que se ha establecido la planificación inicial, es momento de ir cumpliendo con los *sprints* establecidos. A continuación, se detalla como se han organizado los diferentes *sprints* durante el desarrollo del proyecto.

- **Sprint 1. (02/11/2020 - 20/11/2020)**
  Este primer *sprint* se ha centrado en la creación del esqueleto del proyecto, especialmente de la parte del servidor y la base de datos. Además, el objetivo era completar una versión inicial de la aplicación con funcionalidades muy básicas relativas a la gestión de usuarios. En resumen, este *sprint* incluye las siguientes tareas:

  - Generación del esqueleto del proyecto, en el servidor con el generador de Spring (*Spring initializer*) y en el cliente con el generador de Vue a través de su herramienta de línea de comandos (CLI por sus siglas en inglés).
  - Creación de la base de datos.
  - Registrarse en la base de datos.

  - Iniciar sesión.
  - Editar un usuario.
  - Eliminar un usuario.

- **Sprint 2. (20/11/2020 - 10/12/2020)**
  En este *sprint* estaba previsto realizar todo lo referente a la gestión básica de los sitios de interés:

  - Creación del usuario administrador.
  - Crear sitio de interés turístico.
  - Editar sitio de interés turístico.
  - Eliminar sitio de interés turístico.
  - Ver información del sitio.
  - Ver listado de los sitios de interés.

- **Sprint 3. (10/12/2020 - 30/01/2021)**
  En este tercer *sprint* se avanzaría en la parte de editar el perfil de usuario y poder cambiar fotos, tanto del usuario como de los sitios turísticos. Además, se comenzaría con la implementación de los elementos encargados del filtrado de la información disponible. En resumen, los objetivos para este *sprint* eran:

  - Ver perfil del usuario.
  - Cambiar foto del perfil del usuario.
  - Añadir imágenes complementarias en los sitios de interés.
  - Crear buscador de sitios de interés con filtros de categoría.
  - Crear página con las categorías y temas para filtrar los sitios.

- **Sprint 4. (30/01/2021 - 19/02/2021)**
  Durante este *sprint*, aparte de añadir nuevas funcionalidades a la aplicación, también se quería modificar el diseño del cliente para hacerlo más simple y cuidado. Para ello, se decidió utilizar TailwindCSS que proporciona un forma moderna e intuitiva para trabajar con CSS. En este *sprint* se programaron, por tanto, las siguientes tareas:

  - Dar de alta un sitio cultural cubriendo todos los campos importantes.
  - Marcar un sitio de interés como favorito.
  - Marcar un sitio de interés como pendiente de visitar.
  - Ver listado de sitios favoritos.

  - Ver listado de sitios pendientes de visitar.
  - Elaborar la página de últimos sitios introducidos en la aplicación.

- **Sprint 5. (19/02/2021 - 08/02/2021)**
  Este *sprint* estaba pensado para ser dedicado principalmente a la creación del mapa, entender como funciona Hibernate Spatial y como guardar los datos utilizando PostGIS. En concreto, se abordarían los siguientes puntos:

  - Puntuar un sitio de interés para crear comunidad entre los usuarios.
  - Elaboración del mapa.
  - Manejar las coordenadas y ubicarlas correctamente.
  - Control y agrupamiento para que no exista una nube poco manejable de marcadores en el mapa.
  - Elaboración de la memoria del proyecto.

4.2.3 Estimación de costes del proyecto

Para realizar una estimación precisa del coste asociado al desarrollo de un proyecto de este tipo, es importante considerar los recursos necesarios para llevarlo a cabo y los gastos asociados a cada uno de ellos. En este caso, se distinguen dos tipos de recursos:

- **Recursos humanos:** Dentro de los recursos humanos, se pueden considerar básicamente dos roles: el de director de proyecto y el de analista/programador. En el primer caso, dos personas han desempeñado este rol (los tutores del trabajo) mientras que, en el segundo caso, este rol ha sido desempeñado por el autor del proyecto. Dentro del contexto de la metodología Scrum, los directores del proyecto han actuado como *Product Owner* y como *Scrum Master*, mientras que el analista/programador conformaría básicamente el equipo Scrum para el desarrollo del proyecto.

- **Recursos técnicos:** Al considerar soluciones de código abierto, las herramientas software utilizadas no han supuesto un coste adicional en el proyecto. Además, todo el desarrollo del proyecto se ha realizado utilizando el ordenador personal del estudiante. De esta forma, los recursos hardware necesarios ya estaban disponibles y tampoco han supuesto un gasto adicional.

De esta forma, el coste total del proyecto se corresponde exclusivamente con los gastos asociados a los salarios de los recursos humanos empleados en el proyecto. Para determinar estos costes, se ha empleado el Boletín Oficial del Estado [52] del 18 de octubre de 2019, donde se especifica que un analista-programador recibe un salario mensual de 1.712,42 €/mes, lo que

equivale a 26.323,57 €/anuales (mes x 14) asumiendo una jornada laboral anual de 1.792 horas. Teniendo en cuenta que el autor del trabajo empleó una media de 3 horas diarias durante algo más de 3 meses, el número total de horas dedicadas al desarrollo del proyecto es de aproximadamente 315 horas, lo que correspondería a un salario de 4.627,20 €.
Por otro lado, los directores emplearon una media de 2 horas para cada reunión al final de cada *sprint*, más el tiempo extra de las reuniones iniciales y revisando la memoria, hacen un total de 60 horas (30 horas cada uno). Teniendo en cuenta que el salario medio de un jefe de proyecto está sobre los 35000 €/anuales, esto correspondería a un salario de 1171,82 €.
Todo esto hace un total de **5.508,56 €**, que refleja el coste total al final del desarrollo del proyecto.

Recurso humano	Horas	Coste por hora	Coste total
Óscar Fresnedo Arias	30	19,53 €	585,91 €
José Pablo González Coma	30	19,53 €	585,91 €
Junior Paradelo Álvarez	315	14,69 €	4.627,20 €
		Total	**5799,02 €**

Cuadro 4.1: Tabla con un resumen de los costes del proyecto.

Capítulo 5

Análisis

En esta sección se presentarán los actores y los requisitos del proyecto de manera general (tanto funcionales como no funcionales), además de la descripción final del *Product Backlog*, apoyándose en las historias de usuario.

5.1 Actores

Un actor es un usuario que interactúa con la aplicación. En este caso, se han distinguido tres tipos de actores:

- **Usuario anónimo/invitado:** Este tipo de usuario podrá visualizar la página principal, realizar búsquedas de sitios de interés filtrando por categoría, ver los últimos sitios introducidos en la aplicación, navegar por el mapa interactivo y ver los detalles específicos de cada sitio cultural o histórico. Además, podrá acceder al formulario de registro de la aplicación así como al de inicio de sesión.

- **Usuario registrado (*user*):** Es un usuario que ya ha creado una cuenta en la aplicación. Tendrá su propio perfil, en el cual podrá editar sus datos, y podrá guardar sitios en su lista de favoritos o en su lista de sitios pendientes a visitar. Podrá también valorar y puntuar un sitio de interés, así como dejar un comentario en el mismo.

- **Usuario administrador (*admin*):** Además de hacer todo lo que puede hacer un usuario registrado, será el encargado de gestionar la inserción de nuevos sitios culturales o históricos en la aplicación. Tiene permisos para poder eliminar usuarios o sitios de interés. Además de esto puede editar la información de los sitios ya creados.

5.2 Requisitos

En esta sección se exponen tanto los requisitos funcionales como los no funcionales, y se listarán las historias de usuario que se han tenido en cuenta a la hora de desarrollar la aplicación.

5.2.1 Requisitos funcionales

Los requisitos prioritarios que se acordaron en el análisis preliminar de la aplicación fueron los siguientes:

- **Autenticación de usuario:** Un usuario anónimo podrá registrarse en la aplicación e iniciar sesión. No es obligatorio que el usuario se autentifique para visualizar los sitios de interés, ni para realizar búsquedas, pero sí para guardar sitios en las listas de favoritos y pendientes, así como para puntuar, comentar y valorar un sitio.
- **Recuperación de contraseña:** En caso de que un usuario se haya olvidado de su contraseña, se le dará la opción de recuperarla e introducir una nueva contraseña proporcionando el nombre de usuario y el correo electrónico.
- **Gestión de usuario *(user)*:** Una vez que el usuario se haya registrado, éste podrá:
  - Visualizar su perfil.
  - Cambiar la foto de perfil.
  - Editar el propio perfil del usuario.
  - Eliminar el perfil del usuario.
  - Guardar cualquier sitio en su lista de sitios favoritos.
  - Guardar cualquier sitio en su lista de sitios pendientes de visitar.
  - Puntuar cualquier sitio que guarde en la lista de favoritos.
  - Comentar en los detalles específicos de un sitio de interés.
- **Gestión de usuario *(admin)*:** Además de las funcionalidades de un usuario con rol *user*, el usuario con rol *admin* podrá:
  - Dar de alta un nuevo sitio de interés en la aplicación.
  - Editar la información de un sitio ya creado.
  - Dar de baja un sitio de interés en la aplicación.
  - Eliminar un usuario de la aplicación.

- **Funcionalidades específicas:** Se muestran las funcionalidades que se han establecido que tiene que proporcionar la aplicación:
  - Existencia de un buscador con diferentes tipos de filtros para mostrar los sitios que encajen con la búsqueda del usuario.
  - Posibilidad de ordenar los sitios filtrados de acuerdo a diferentes criterios: si se puede ir con niños, si se puede ir en automóvil, etc.
  - Existencia de una página para visualizar los últimos sitios añadidos en la aplicación, pudiendo filtrar por categoría.
  - Existencia de un mapa interactivo donde se muestra la ubicación física de los sitios de interés y desde donde se puede ver, además, la ubicación del punto de acceso para un determinado sitio.
  - Posibilidad de consultar la información detallada de cada sitio de interés desde diferentes zonas de la aplicación.

5.2.2 Requisitos no funcionales

En cuanto a requisitos no funcionales, se ha establecido que la aplicación debe ser:

- **Segura:** Durante el registro del usuario se deberá cifrar la contraseña, aunque habrá una opción para visualizar la contraseña que el usuario está introduciendo. Para mayor seguridad, la contraseña se guardará cifrada en base de datos. Además, debería mantener la privacidad de los usuarios registrados sobre sus perfiles, listas de favoritos y pendientes de visitar, etc.
- **Intuitiva y fácil de usar:** La aplicación deberá mantener un estilo amigable, sencillo y elegante, que favorezca su uso y que sea intuitiva para todo tipo de usuarios.
- **Diseño web adaptable:** La aplicación deberá adaptar la apariencia al dispositivo que se esté utilizando para permitir la correcta visualización de las páginas de la misma.
- **Escalabilidad:** El mapa interactivo de la aplicación deberá ser fluido aunque tenga una cantidad elevada de puntos cargados sobre él.

5.2.3 Pila de producto *(Product Backlog)*

Una vez recogidos los requisitos descritos en las secciones anteriores, se ha entrado en detalle en cada uno de ellos para reflejarlos en una serie de historias de usuario, las cuales se exponen a continuación.

1. Si es un usuario anónimo en la aplicación, debe poder acceder a la página principal para registrarse. Para registrarse, deberá introducir su nombre, apellidos, un login de usuario con el que poder acceder después, una contraseña y un correo electrónico. Puede introducir una imagen de perfil en ese momento si lo desea.

2. Al registrarse en la aplicación, ésta tiene que guardar la fecha de registro de forma automática en la base de datos para mantener una trazabilidad de los datos.

3. Un usuario anónimo podrá acceder al listado de sitios de interés, y podrá ver el perfil de cualquier sitio y toda su información detallada.

4. Un usuario anónimo podrá hacer búsquedas sobre los sitios de interés, pudiendo añadir además algún filtro con las categorías que decida en ese momento.

5. Un usuario podrá ordenar los sitios obtenidos en la búsquedas por diferentes criterios que le ayuden a seleccionar de forma eficiente futuros destinos.

6. Cualquier usuario podrá visualizar el apartado de últimos sitios introducidos en la aplicación para ver novedades que le resulten interesantes.

7. Un usuario registrado podrá iniciar sesión con su nombre de usuario y contraseña, además de poder cerrar sesión, teniendo que volver a introducir las credenciales si quiere volver a acceder.

8. Cualquier usuario podrá acceder al mapa de localización de sitios culturales o históricos. Este mapa debe mostrar la localización física de los sitios de interés con sus correspondientes filtros. Además, si es el caso, debe permitir visualizar un sitio de interés seleccionado junto con su punto de acceso a pie.

9. Como usuario registrado con rol *user* en la aplicación, además de poder realizar lo que un usuario anónimo, éste podrá guardar cualquier sitio cultural o histórico en su lista de favoritos, o de pendientes de visitar.

10. Al ser usuario registrado *(user)*, podrá acceder a su perfil de usuario. Dentro de su perfil de usuario podrá acceder al listado de sitios favoritos y sitios pendientes de descubrir.

11. Al ser usuario registrado *(user)*, dentro del perfil del propio usuario, podrá modificar su información (nombre, apellidos, correo electrónico e imagen de perfil).

12. Como usuario registrado (*user*), podrá eliminar su perfil de la aplicación.

13. Al ser un usuario registrado (*user*), se le dará la opción de puntuar un sitio cultural. La puntuación máxima está marcada como cinco estrellas.

14. Como usuario administrador *(admin)*, además de las posibilidades de un usuario registrado, éste podrá acceder a otras funcionalidades como el formulario de creación de un sitio de interés. Para ello, se deberá introducir obligatoriamente los siguientes datos: nombre, provincia, categoría, coordenadas del lugar, breve descripción del lugar, un encabezado para los detalles del sitio, un resumen sobre el sitio y si existe alguna restricción sobre el sitio. Otros datos que se podrían introducir, aunque no de manera obligatoria, serían: el municipio, coordenadas del aparcamiento disponible, indicaciones de acceso, posibilidad de ir con niños, posibilidad de ir en automóvil. Además de esto se podrá añadir una imagen principal del sitio y varias imágenes complementarias que se visualizarán de forma agradable en los detalles de cada sitio.

15. Como usuario administrador, dentro de un sitio de interés, aparecerá la opción de eliminarlo. Esta acción supone inhabilitar toda la información relacionada con ese lugar.

16. Un usuario administrador podrá eliminar a cualquier usuario de la aplicación.

17. Un usuario administrador podrá editar la información correspondiente a cualquier sitio de interés creado en la aplicación.

5.3 Maquetas de los componentes

A partir del análisis de las funcionalidades de la aplicación, se ha realizado una primera aproximación de los elementos necesarios en la interfaz para dar soporte a dichas funcionalidades. En esta sección, se presenta una descripción de alto nivel de la interfaz de usuario de la aplicación, enumerando las pantallas, describiendo su estructura general e indicando cómo se realiza la navegación entre ellas. Para ello, se han utilizado unos *mockups* que se han creado en la etapa inicial del proyecto. Este tipo de maquetas se realizan antes de comenzar el desarrollo de la aplicación para tener una idea inicial y no tener que estructurar todo sobre la marcha.

La figura 5.1 muestra lo que sería la pantalla principal de la aplicación, la cual se puede visualizar siendo un usuario anónimo. En ella, encontraríamos una breve descripción de la página y unos accesos directos a las diferentes funcionalidades de la misma: ir al buscador, ver las novedades en los últimos sitios introducidos en la aplicación, ir a la zona de las categorías e ir al mapa interactivo.

La figura 5.2 muestra la pantalla que aparecería al hacer clic sobre el botón de iniciar sesión. En esta pantalla, se muestra un formulario con los campos necesarios para crear una cuenta en la aplicación. Una vez que se haya hecho una cuenta, el usuario podrá acceder a la aplicación como un usuario registrado, y podrá ver su perfil con la lista de sitios favoritos y

Figura 5.1: Página principal - Usuario anónimo

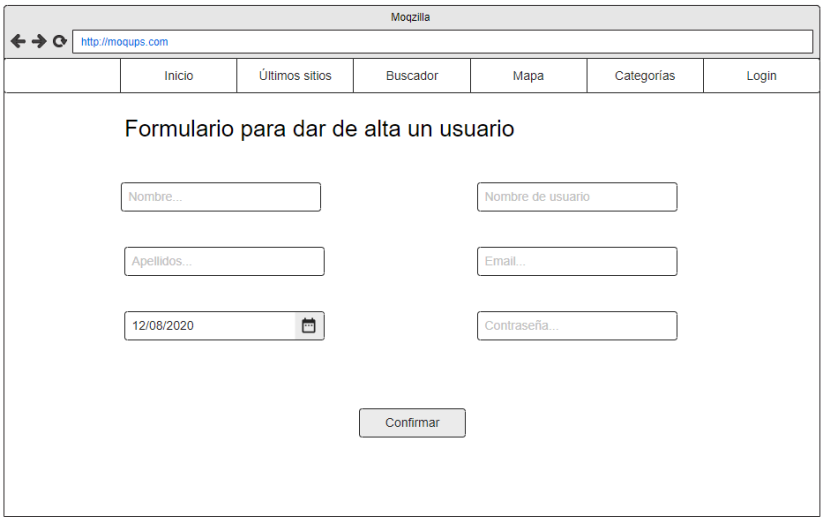

Figura 5.2: Página de registro - Usuario anónimo

sitios pendientes de visitar (como se puede ver en la figura 5.3).

Siendo un usuario registrado, la vista general de un sitio sería tal y como se muestra en la figura 5.4, donde se podrá ver la información de cada sitio, fotos añadidas para ese sitio y su valoración media.

El buscador general para encontrar un sitio se muestra en la figura 5.5. Como se puede apreciar, existen diferentes filtros de categoría para acotar la búsqueda.

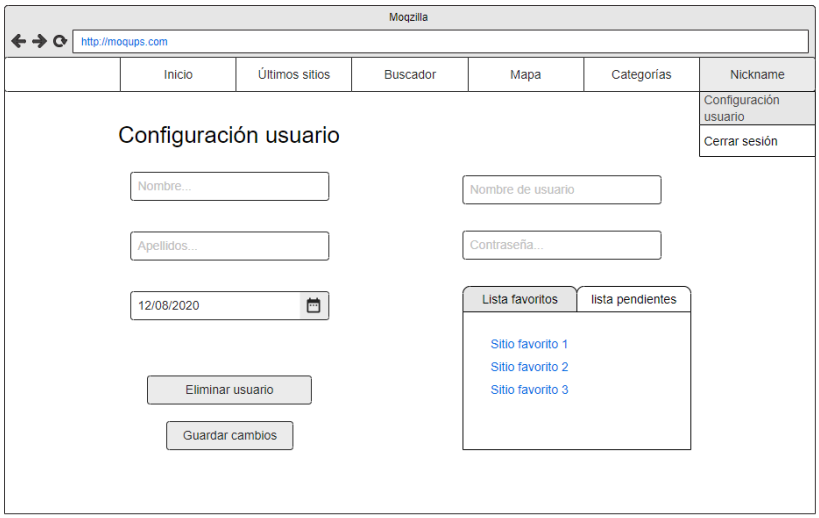

Figura 5.3: Página principal para Usuario registrado

Figura 5.4: Página con información sobre sitio de interés - Usuario registrado

En la página correspondiente a los últimos sitios (figura 5.6), podemos ver las novedades y últimos sitios introducidos en la aplicación ordenados por fecha.

Además de esto, tenemos la funcionalidad del mapa interactivo (figura 5.7), en el que se muestra el mapa con los puntos turísticos de cada zona. Todo ello con una leyenda indicando todo lo que se visualiza.

Figura 5.5: Página del buscador - Usuario anónimo

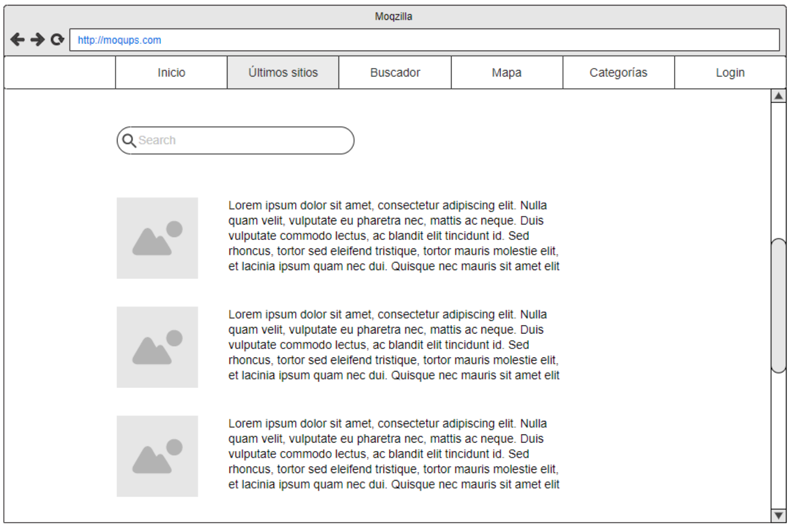

Figura 5.6: Página de últimos sitios - Usuario anónimo

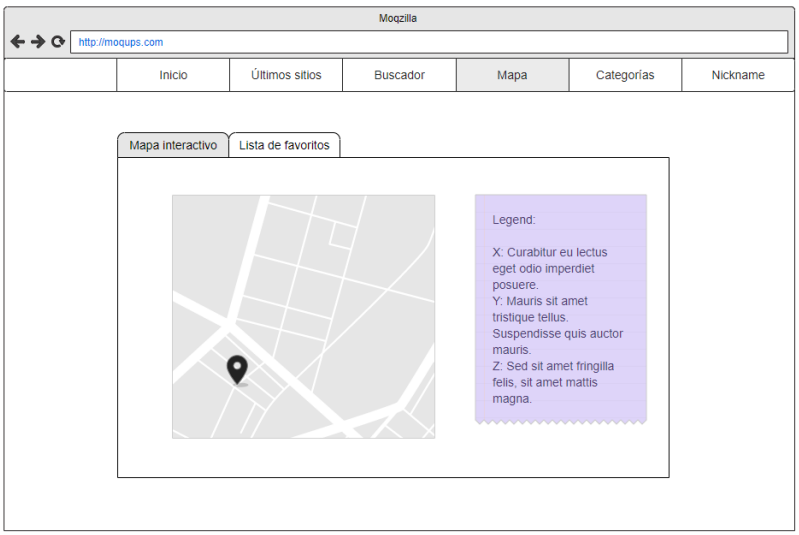

Figura 5.7: Página del mapa - Usuario registrado

5.4 Arquitectura del sistema

En este apartado se detalla la descripción de alto nivel de la arquitectura del sistema. Se diferencia entre parte correspondiente al servidor *(backend)* y la parte correspondiente al cliente *(frontend)*.
Como se puede ver en la figura 5.8, la aplicación sigue una arquitectura cliente-servidor, donde el cliente se comunicará con el servidor mediante servicios REST. Con esto se consigue que las capas del servidor y del cliente estén totalmente independizadas, lo que facilitará su evolución por separado y su flexibilidad a la hora de reutilizarse.
En particular, se ha decidido utilizar una arquitectura basada en el patrón *Model-view-controller (MVC)* [53]. Al utilizar este patrón de diseño, se obtienen las siguientes ventajas:

- La clara separación de responsabilidades impuesta por el uso del patrón MVC hace que los componentes de nuestra aplicación tengan sus misiones bien definidas. Por lo tanto, nuestro sistema será más limpio, simple, más fácilmente mantenible y, por tanto, más robusto.
- Al estar separado en tres partes tan diferenciadas, diferentes programadores podrían ocuparse de cada parte en paralelo.
- Se podrían crear múltiples vistas a partir del mismo modelo, pudiendo reaprovechar mucho mejor los desarrollos y asegurando consistencia entre ellas.
- Facilidad para realización de pruebas unitarias.

Por tanto de esta forma, se consigue que exista abstracción, escalabilidad y aislamiento a cambios ya que solo se conoce la capa inmediatamente inferior, por lo que se pueden realizar cambios en cada capa que no afectan a las capas superiores. Esto también permite que haya una alta cohesión ya que permite la agrupación lógica de acciones en un mismo controlador, que lo hace más fácil de entender y reutilizar.

5.4.1 Cliente

El cliente es el encargado de definir la interfaz gráfica para el usuario y su interacción con dicha interfaz, de forma que el cliente puede enviar información al servidor gracias a la interfaz de tipo REST que el servidor le proporciona. El cliente sólo conoce esta interfaz de todo el servidor por lo cual es de **bajo acoplamiento**. Además, el cliente debe presentar de forma correcta la información que se obtiene como resultado de la interacción del usuario con la aplicación.

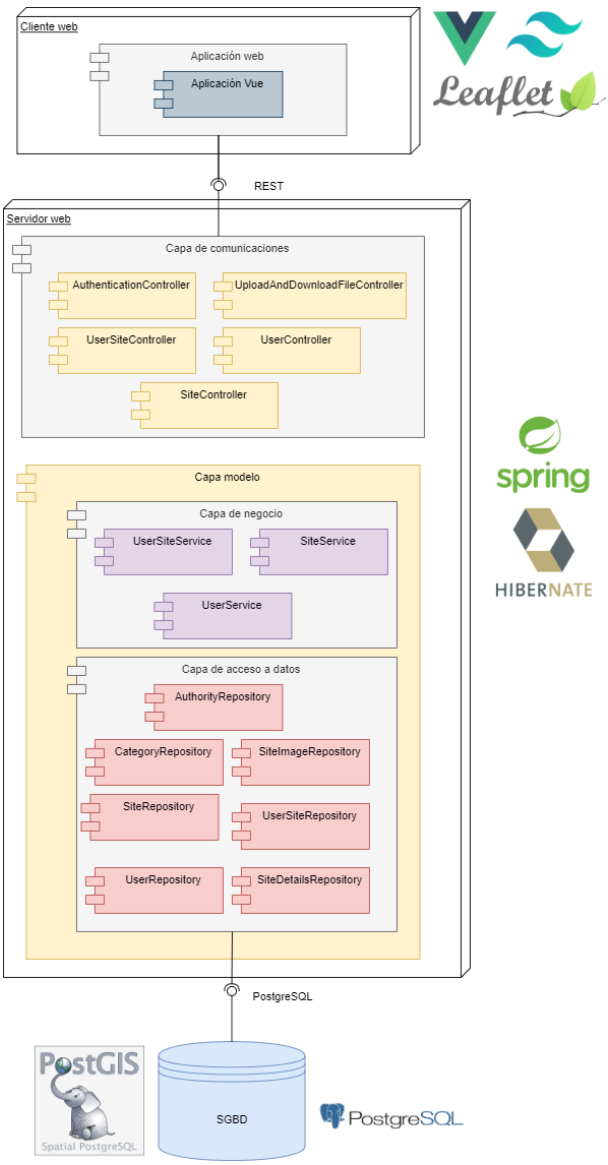

Figura 5.8: Arquitectura del sistema

El cliente de este proyecto está desarrollado siguiendo una arquitectura *Model-view-viewmodel (MVVM)* [54] utilizando el *framework* de Vue.js.

5.4.2 Servidor

El servidor está formado por una arquitectura en capas en la que cada capa solo trata la lógica específica de esa capa, sin saber como están implementadas las demás. En concreto, se definen las siguientes tres capas:

- **Capa de acceso a datos *(Repository):*** Esta capa es la encargada de la comunicación

con la base de datos.

- **Capa modelo de la lógica de negocio *(Service):*** Se implementa la lógica de las operaciones necesarias para el correcto funcionamiento de la aplicación.
- **Capa de comunicaciones *(Controller):*** En esta capa se implementan los puntos de acceso (*endpoints*) que pueden ser invocados de forma externa a la aplicación (por el cliente en este caso). A través de estos puntos de acceso se reciben las peticiones del cliente, que se pasan a la capa de modelo de negocio para poder realizarlas. Para esta conexión se ha utilizado un servicio tipo REST, que es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web basado en el protocolo HTTP, donde se envían peticiones a una URI utilizando los métodos disponibles, es decir, métodos GET, PUT, POST, DELETE, etc.

Capítulo 6

Diseño

En este capítulo se describirá en detalle la arquitectura tecnológica utilizada en la aplicación que hemos visto en la figura 5.8 de la sección anterior. Se hará una distinción entre las 3 capas más relevantes de la arquitectura: la parte de almacenamiento de datos, la parte del servidor y la parte del cliente.
Las principales características de estos tres componentes son (ver figura 5.8):

- Para el **almacenamiento de datos**, se ha utilizado PostgreSQL, un sistema de gestión de bases de datos relacional y orientado a objetos. El motivo principal de la elección de este gestor es que incluye PostGIS, una extensión que convierte el sistema en una base de datos espacial mediante la adición de tipos de datos espaciales que dan soporte a la necesidad de almacenar las coordenadas de los sitios de interés turístico correctamente.

- La capa de almacenamiento de datos se comunicará con la parte del **servidor** a través de clases que utilizan el patrón *Data Access Object* (DAO). A su vez, esta capa de acceso de datos se comunica con los servicios implementados con Spring e Hibernate, donde se llevarán a cabo las distintas operaciones permitidas por la aplicación.

- Esta última capa expondrá un servicio REST para asegurar la comunicación con el **cliente**, que se corresponde con una aplicación web implementada con Vue.js, un *framework* de JavaScript de código abierto para desarrollar interfaces de usuario mediante componentes. El cliente web usará TailwindCSS para el diseño de las vistas de las pantallas y Leaflet, una librería de JavaScript para visualizar mapas e interactuar con ellos.

6.1 Diseño de la aplicación

En esta sección se describirá de manera detallada las partes que conforman la aplicación y cómo se ha implementado la seguridad en la misma.

6.1.1 Base de datos

En esta sección se explica la capa modelo del servidor. A las clases del modelo se les ha puesto la etiqueta @Entity para definirlas. De esta forma, cada atributo de cada entidad tendrá sus correspondientes métodos *getter* y *setters*, y los constructores necesarios.
Cabe destacar que, para agilizar la escritura del código se ha utilizado Lombok [55], que es una biblioteca de Java que nos permite mediante anotaciones dejar el código más limpio y de fácil visibilidad para el usuario programador.
En la figura 6.1 se muestra el diagrama con las entidades necesarias en la aplicación y sus correspondientes relaciones.

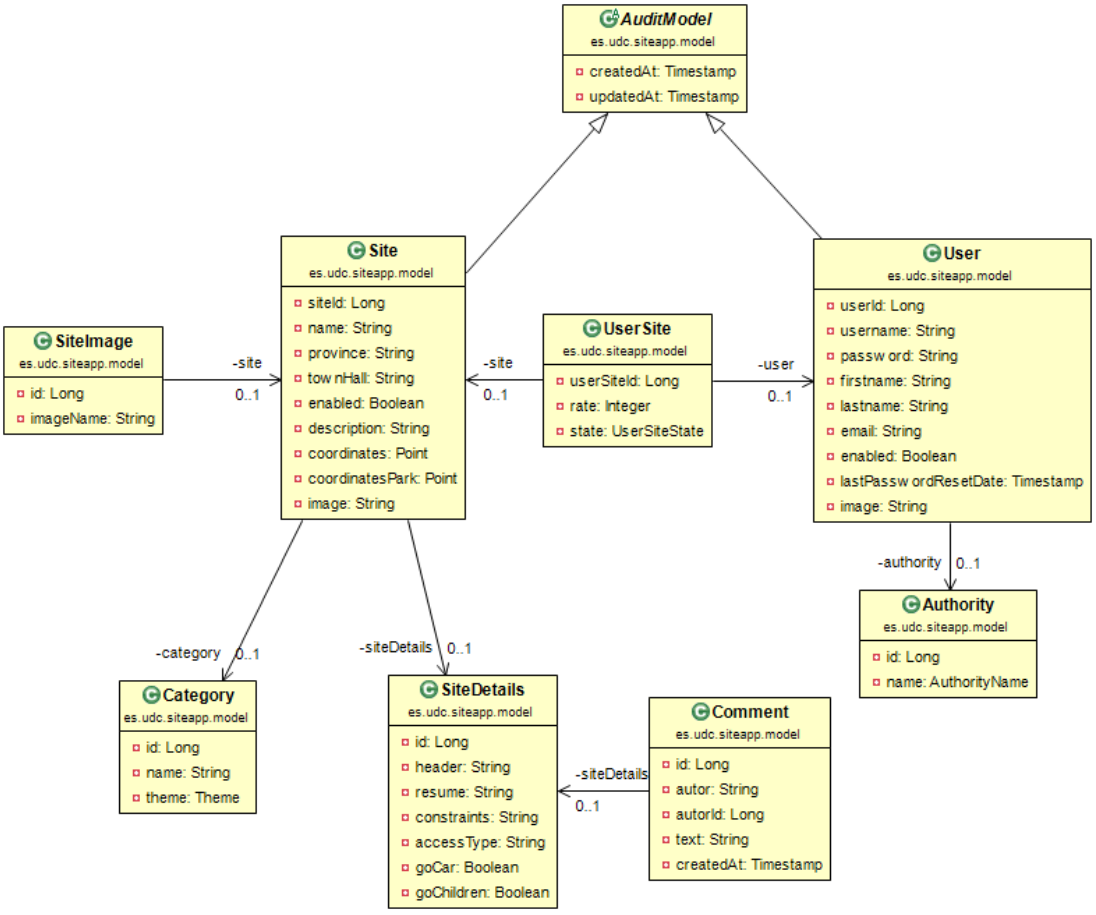

Figura 6.1: Entidades de la aplicación

Como se puede apreciar, existe una entidad denominada AuditModel de la cual extienden sus propiedades de fecha de creación y fecha de actualización las entidades principales Site, que define las propiedades de un sitio de interés, y User, que define las propiedades de

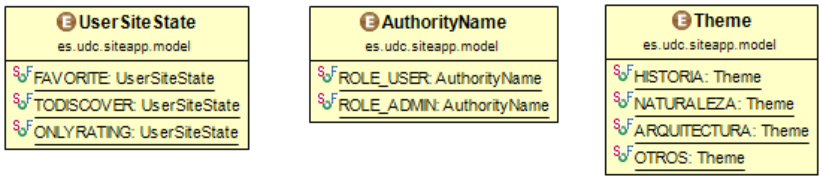

Figura 6.2: Enumerados de la aplicación

un usuario de la aplicación. Como entidades secundarias tendríamos las entidades Authority, Category, SiteDetails, Comment y SiteImage que complementan a las entidades principales. Además de esto, tenemos la entidad UserSite que surge como relación entre un usuario y un sitio de interés.
Por otro lado, cabe destacar que en la aplicación existen tres enumerados: *AuthorityName, Theme y UserSiteState*. El primero de ellos, *AuthorityName*, representa el rol de autoridad para un usuario y está relacionado directamente con la clase *Authority*. El segundo de ellos, *Theme* representa los diferentes temas que existen en la aplicación. Y el tercero de ellos, *(UserSiteState)*, indica el estado de la relación entre usuarios y sitios de interés.

Modelo conceptual de datos

En esta sección, se describirán en detalle las diferentes entidades que se almacenarán de forma persistente en la base de datos de la aplicación con sus correspondientes atributos.

- **AuditModel:** Superclase de la que provienen los usuarios y los sitios de interés, y que proporciona la fechas de creación y actualización de los mismos. Cuenta con los siguientes atributos:
  - createdAt: fecha de creación del usuario o del sitio de interés.
  - updatedAt: fecha de actualización de los datos.
- **User:** Representa a un usuario de la aplicación. Cuenta con los siguientes atributos:
  - userId: identificador de un usuario. Se genera mediante una secuencia.
  - username: nombre que el usuario tendrá en la aplicación y con el que podrá autenticarse.
  - password: contraseña para acceder a la aplicación.
  - firstname: nombre propio del usuario.
  - lastname: apellidos del usuario.
  - email: correo electrónico del usuario.
  - enabled: *flag* que indica si el usuario está des/habilitado.

  - lastPasswordResetDate: fecha de la última vez que cambió la contraseña el usuario.
  - authority: rol de autoridad del usuario.
  - image: imagen principal del usuario, guardada en base de datos en formato base64.

- **Authority:** Representa los diferentes roles de los usuarios y tiene los siguientes atributos:
  - id: identificador del rol. Se genera mediante una secuencia.
  - name: nombre del rol.

- **AuthorityName:** Representa los nombres de los roles que existen en la aplicación. Tiene los siguientes atributos:
  - *USER*: Rol para un usuario registrado.
  - *ADMIN*: Rol para un usuario administrador.

- **Site:** Representa a un sitio turístico o cultural en la aplicación. Guarda información para los siguientes campos:
  - siteId: identificador de un sitio cultural. Se genera mediante una secuencia.
  - name: nombre del sitio cultural o turístico.
  - province: provincia en la que se encuentra el sitio.
  - townHall: ayuntamiento en donde se encuentra el sitio.
  - enabled: *flag* que indica si el sitio está des/habilitado.
  - category: categoría a la que pertenece el sitio.
  - description: breve descripción sobre el sitio.
  - siteDetails: detalles específicos de cada sitio de innterés.
  - coordinates: coordenadas con la ubicación del sitio.
  - coordinatesPark: coordenadas con la ubicación del aparcamiento disponible para el sitio.
  - image: imagen principal del sitio cultural, guardada en base de datos en formato base64.

- **SiteDetails:** Representa los detalles específicos de cada sitio. Cuenta con los siguientes atributos:
  - id: identificador de un detalle de un sitio. Se genera mediante una secuencia.
  - header: información principal y más importante de un sitio.

  - resume: información extensa de un sitio.
  - constraints: información adicional de un sitio.
  - accessType: indicaciones sobre el tipo de acceso de un sitio.
  - goCar: *flag* que indica si se puede llegar en automóvil al sitio.
  - goChildren: *flag* que indica si es posible/recomendable ir con niños al sitio.

- **Comment:** Representa los comentarios que cada usuario deja en un sitio. Para cada comentario, se guarda la siguiente información:
  - id: identificador único de un comentario.
  - autor: nombre del autor que realiza el comentario.
  - autorId: identificador del autor que realiza el comentario.
  - text: texto que contiene el comentario que realiza el autor.
  - createdAt: fecha y hora en la que se realiza el comentario.
  - siteDetails: identificador los detalles de sitio en el que se encuentra el comentario.

- **SiteImage:** Representa las imágenes complementarias de un sitio. Tiene los siguientes campos:
  - id: identificador de la imagen complementaria. Se genera mediante una secuencia.
  - imageName: nombre de la imagen complementaria.
  - site: sitio al que pertenece la imagen.

- **UserSite:** Representa la entidad intermedia que relaciona a los usuarios con un sitio de interés, permitiendo saber la puntuación y el estado del usuario sobre cada sitio. Por tanto, cuenta con los siguientes atributos:
  - userSiteid: identificador. Se genera mediante una secuencia.
  - site: sitio de interés concreto.
  - user: usuario en concreto.
  - rate: puntuación que el usuario valora sobre el sitio cultural.
  - state: estado que le ha asignado el usuario al sitio.

- **UserSiteState:** Representa los 3 estados posibles que un usuario puede marcar para un sitio en la aplicación, es decir:
  - *FAVORITE:* estado que indica que el sitio de interés está en la lista de favoritos del usuario.

  - *TODISCOVER:* estado que indica que el sitio está en la lista de pendientes de visitar del usuario.
  - *ONLYRATING:* estado que indica que el sitio de interés ha sido puntuado por un usuario.

- **Category:** Representa las categorías disponibles para cada sitio cultural. Tiene los siguientes atributos:
  - id: identificador. Se genera mediante una secuencia.
  - name: nombre de la categoría.
  - theme: tema común al que se le ha asignado la categoría.

- **Theme:** Representa los temas comunes en los que se agrupan las categorías que existen en la aplicación:
  - *HISTORIA:* Engloba a todas las categorías que tienen un componente histórico.
  - *NATURALEZA:* Engloba a todas las categorías que tienen un componente que tiene que ver con la naturaleza.
  - *ARQUITECTURA:* Engloba a todas las categorías que tienen un componente arquitectónico.
  - *OTROS:* Engloba a cualquier categoría que no esté definida entre las otras.

6.1.2 Servidor

En esta sección se explicarán las diferentes capas que forman parte del *backend* de la aplicación.
La tecnología utilizada para su desarrollo ha sido **Java** *(AdoptOpenJDK)* en su versión 8. Además, se ha utilizado **Spring Boot**, un *framework* para la inyección de dependencias, e **Hibernate** para establecer una comunicación más óptima con la base de datos. Podemos ver en la figura 6.3 la estructura básica del servidor.
Como ya hemos visto, la parte del servidor consta de 3 capas claramente diferenciadas: capa de acceso a datos, capa de modelo de negocio y la capa de comunicaciones. Cada una de estas capas será desglosada en las siguientes subsecciones.

Capa de acceso a datos *(Repository)*

Esta capa es la que se encarga del acceso y comunicación con los datos almacenados en la base de datos. Utiliza el patrón DAO para comunicarse con el gestor de almacenamiento.

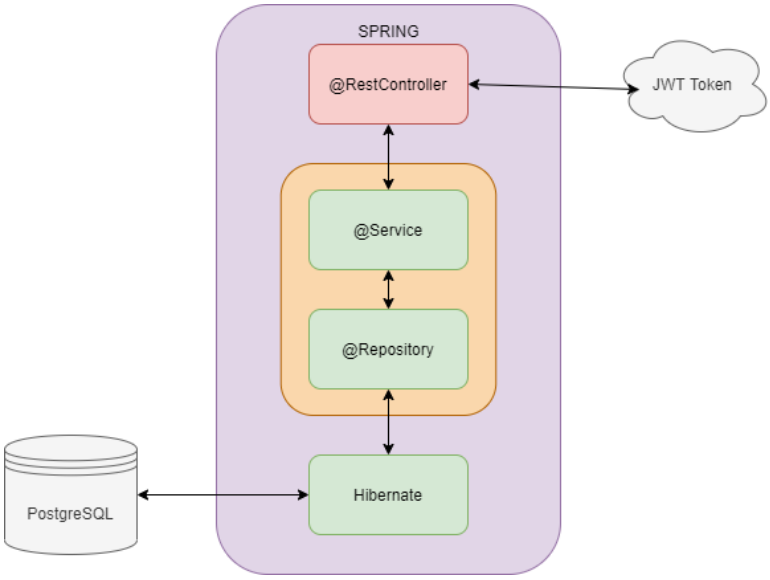

Figura 6.3: Estructura básica del servidor

Este patrón propone separar por completo la lógica de negocio de la lógica de acceso a datos proporcionando los métodos necesarios para insertar, actualizar, eliminar y consultar la información almacenada en la base de datos.
Existirá un repositorio por cada entidad, cada uno con su interfaz y su correspondiente implementación. En cada uno de ellos, se utilizarán los métodos de crear-leer-actualizar-eliminar (CRUD por sus siglas en inglés) proporcionados por *Hibernate*, además de operaciones de recuperación de información creadas por el autor utilizando el lenguaje *Hibernate Query Language (HQL)*. En la figura 6.4 se puede apreciar un ejemplo de consulta utilizando este lenguaje.

```
@Modifying
@Query("from UserSite where user_id = :userId and state = :state")
List<UserSite> findByState(@Param("userId") Long userId, @Param("state") String state);
```

Figura 6.4: Ejemplo HQL

Como podemos ver en la figura 6.5 existen ocho repositorios que extienden del *JpaRepository* y que contienen las operaciones básicas CRUD. Además de esas operaciones, gracias a Hibernate se implementan una serie de operaciones que *JpaRepository* **no proporciona** pero que son necesarias para el funcionamiento de la aplicación. A continuación, se detallan de forma específica estas operaciones para cada repositorio:

1. **UserSiteRepository:**

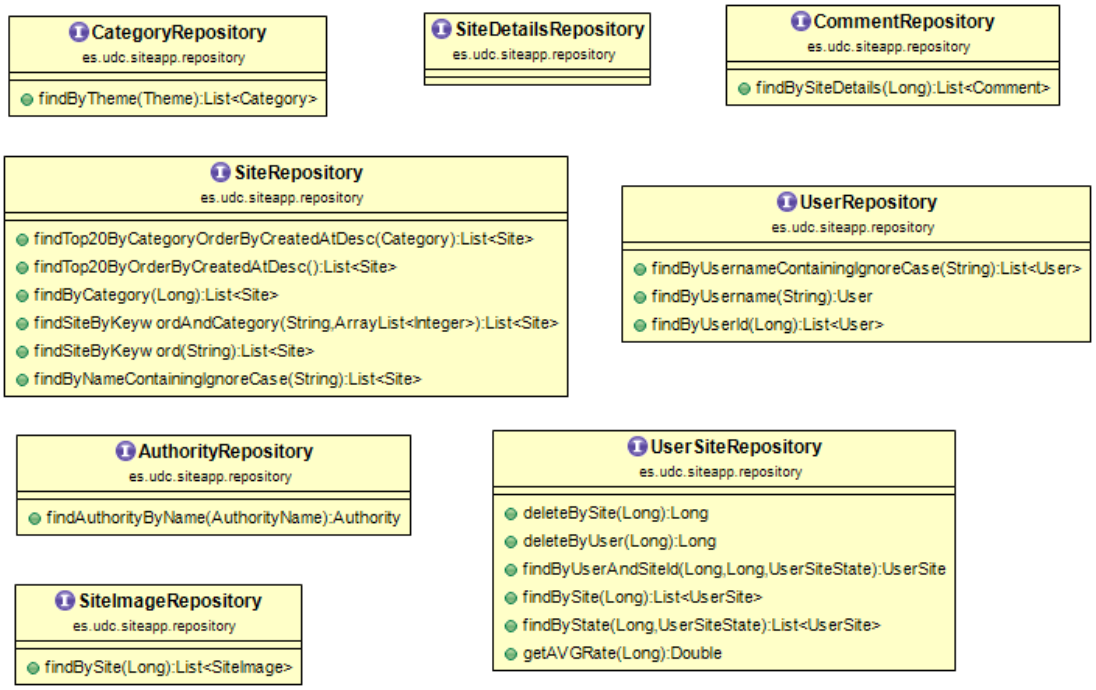

Figura 6.5: Repositorios de la aplicación

- deleteBySite: elimina las entradas asociadas a un sitio a partir del identificador del sitio de interés.
- deleteByUser: elimina las entradas asociadas a un usuario a partir del identificador del usuario.
- findByUserAndSiteId: devuelve un objeto de tipo UserSite a partir del identificador del usuario, identificador del sitio y tipo de estado de la relación.
- findBySite: devuelve una lista de objetos de tipo UserSite a partir del identificador del sitio.
- findByState: devuelve una lista de objetos UserSite a partir del identificador de un usuario e indicando el tipo de estado de la relación.
- getAVGRate: devuelve la media de la puntuación de un sitio determinado a partir de su identificador.

2. **SiteRepository:**

- findByCategory: devuelve una lista de objetos de tipo Site a partir del identificador de la categoría.
- findSiteByKeywordAndCategory: devuelve una lista de objetos de tipo Site que tiene en cuenta las palabras clave que se le pasan y la lista de categorías.

- findSiteByKeyword: devuelve una lista de objetos de tipo Site que tiene en cuenta las palabras clave que se le pasan y sin tener en cuenta ninguna categoría.
- findTop20ByCategoryOrderByCreatedAtDesc: devuelve los últimos veinte sitios introducidos en la aplicación filtrando por categoría.
- findTop20ByOrderByCreatedAtDesc: devuelve los últimos veinte sitios introducidos en la aplicación.
- findByNameContainingIgnoreCase: devuelve los sitios que en su nombre contengan los datos que se le pasan.

3. **SiteImageRepository**

   - findBySite: devuelve la lista de objetos SiteImage a partir del identificador del Site.

4. **UserRepository**

   - findByUsername: devuelve un objeto de tipo User a partir de su campo nombre.
   - findByUserId: devuelve una lista de objetos de tipo User a partir de un identificador. En el caso de nuestra aplicación devolverá un único usuario.
   - findByUsernameContainingIgnoreCase: devuelve una lista de objetos de tipo User que en su nombre contengan los datos que se le pasan.

5. **AuthorityRepository**

   - findAuthorityByName: devuelve un objeto de tipo Authority a partir de su nombre.

6. **CategoryRepository**

   - findByTheme: devuelve una lista de objetos de tipo Category de un tema en concreto.

7. **CommentRepository**

   - findBySiteDetails: devuelve una lista de objetos de tipo Comment de los detalles de un sitio concreto.

Capa de negocio *(Service)*

En esta capa, se implementa la lógica de negocio de la aplicación considerando los requisitos definidos en el análisis previo. Estos servicios se comunicarán con la capa de acceso de datos explicada en el punto anterior para trabajar con los datos que se necesitan en cada

operación. Por otro lado, también se comunicará con los controladores que le solicitan la información requerida por el cliente web y que desencadenan la ejecución de las operaciones correspondientes.
Esta capa hace uso de los denominados *Data Transfer Objects* (DTO) ya que la información no se necesita enviar tal y como está en base de datos, sino que es necesario adaptar o combinar esa información dependiendo de lo que demande el cliente. En la figura 6.6, se muestran los DTO que ha sido necesario implementar en la aplicación.

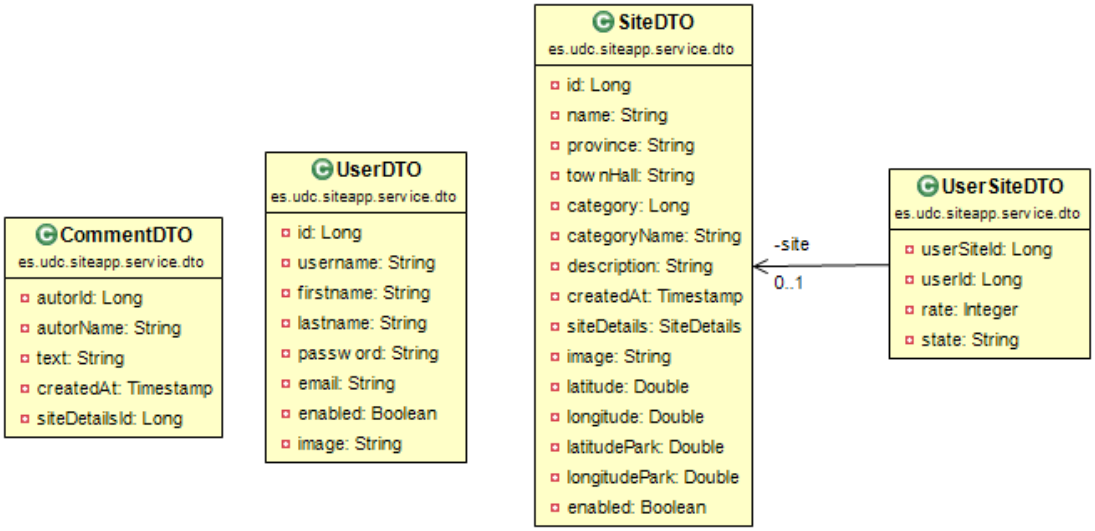

Figura 6.6: DTOs utilizados de la aplicación

Como se puede ver en la figura 6.7, los servicios que se han definido para esta aplicación son cuatro: *UserService, SiteService, UserSiteService* y *FileStorageService*. A continuación, se detallan estos servicios con el conjunto de operaciones que proporcionan:

1. **UserService:** Este servicio implementa los métodos que se utilizan para cada una de las operaciones que se realizan sobre un usuario de la aplicación. En particular, tenemos las siguientes operaciones:

   - findAll: recupera todos los usuarios de la aplicación.
   - getUserById: recupera un usuario concreto de la aplicación por su identificador.
   - getUserByName: recupera un usuario de la aplicación por su nombre de usuario.
   - registerUser: registra un usuario en la aplicación.
   - uploadImage: sirve para establecer una imagen en el perfil del usuario.
   - getImage: sirve para obtener la imagen del perfil de un usuario.

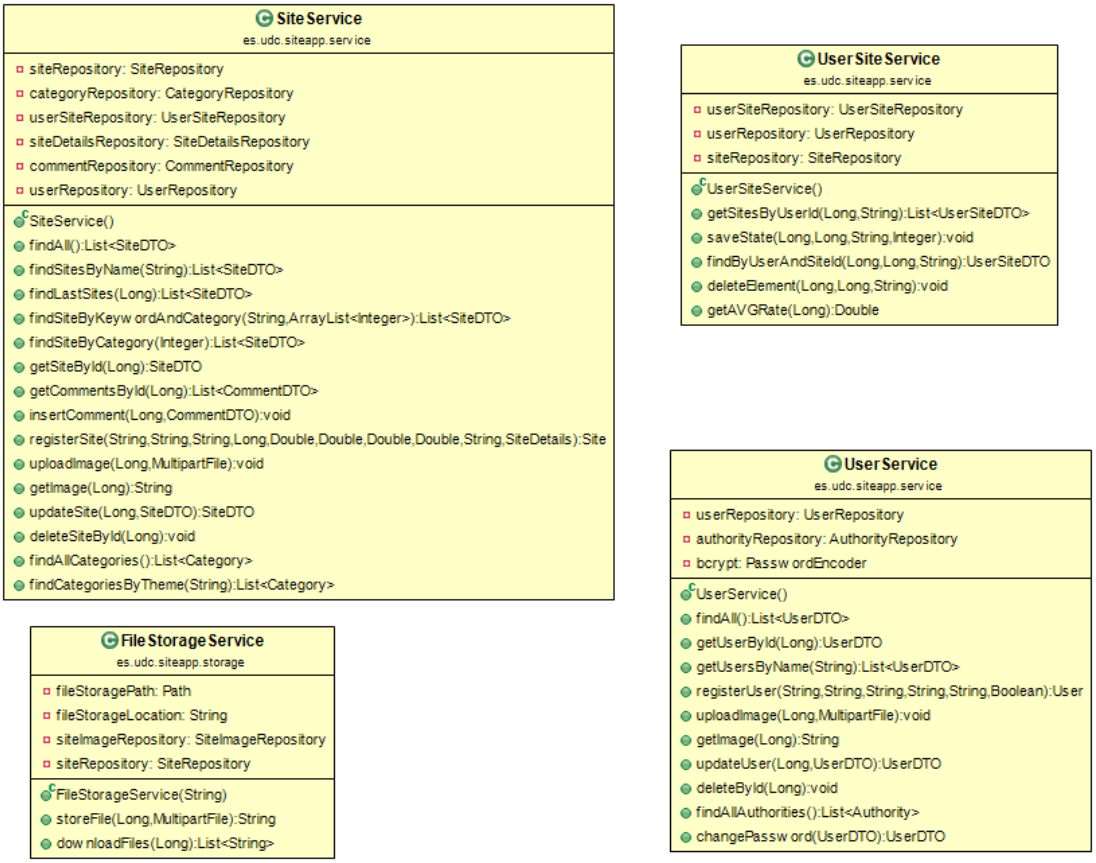

Figura 6.7: Servicios de la aplicación

- updateUser: actualiza los datos permitidos de un usuario (no es posible cambiar su nombre de usuario ni el correo electrónico).
- deleteById: elimina un usuario concreto.
- findAllAuthorities: devuelve los roles de un usuario concreto.
- changePassword: método para cambiar la contraseña en caso de que el usuario se haya olvidado o lo solicite.

2. **SiteService:** Este servicio implementa los métodos que son necesarios para gestionar los sitios de interés. En particular, se definen las siguientes operaciones:

   - findAll: recupera todo el listado de sitios de interés.
   - findSitesByName: recupera el listado de sitios filtrando por nombre.
   - findLastSites: recupera el listado de los últimos sitios introducidos en la aplicación.
   - findSiteByKeywordAndCategory: recupera los sitios que su cuyo nombre y/o categoría corresponden con los que se le pasan.

- findSiteByCategory: recupera todos los sitios de una categoría concreta.
- getSiteById: recupera un sitio de interés a partir de su identificador.
- getCommentsById: obtiene los comentarios de un sitio a partir de su identificador.
- insertComment: inserta un comentario para un determinado sitio a partir del identificador del sitio y de los datos del comentario.
- registerSite: da de alta un nuevo sitio de interés con la información proporcionada.
- uploadImage: se guarda la imagen principal para un sitio.
- getImage: se obtiene la imagen principal de un sitio.
- updateSite: se actualizan los datos de un sitio de interés.
- deleteSiteById: se elimina un sitio.
- findAllCategories: se obtienen todas las categorías existentes en la aplicación.
- findCategoriesByTheme: se obtienen todas las categorías en la aplicación de un tema en concreto.

3. **UserSiteService:** Este servicio implementa los métodos que son necesarios para mantener las relaciones entre usuarios y sitios culturales. Las operaciones proporcionadas por este servicios son las siguientes:

   - getSitesByUserId: obtiene los sitios de interés para un determinado tipo de relación con un usuario (favoritos, pendientes).
   - saveState: almacena una relación entre un sitio de interés y un usuario.
   - findByUserAndSiteId: devuelve una relación concreta, que incluye la puntuación del usuario, pasándole el usuario, sitio y tipo de relación.
   - deleteElement: elimina una relación entre un usuario y un sitio de interés.
   - getAVGRate: obtiene la media de las puntuaciones realizadas para un determinado sitio.

4. **FileStorageService:** Este servicio implementa los métodos que son necesarios para tratar las imágenes complementarias de los sitios de interés. En concreto:

   - storeFile: añade una imagen complementaria para un determinado sitio en el directorio establecido en la configuración.
   - downloadFiles: descarga todas las imágenes complementarias de un sitio de interés a partir de su identificador.

Capa de comunicaciones *(Controller)*

Esta capa se comunicará directamente con la capa de servicios explicada en el apartado anterior, llamando a los métodos apropiados en función de las peticiones del cliente. Para ello, los **controladores** exponen los diferentes puntos de acceso para la conexión con el cliente y redirigen estas peticiones hacia la capa de servicios. En la figura 6.8, se muestran los controladores que se utilizan en el servidor.

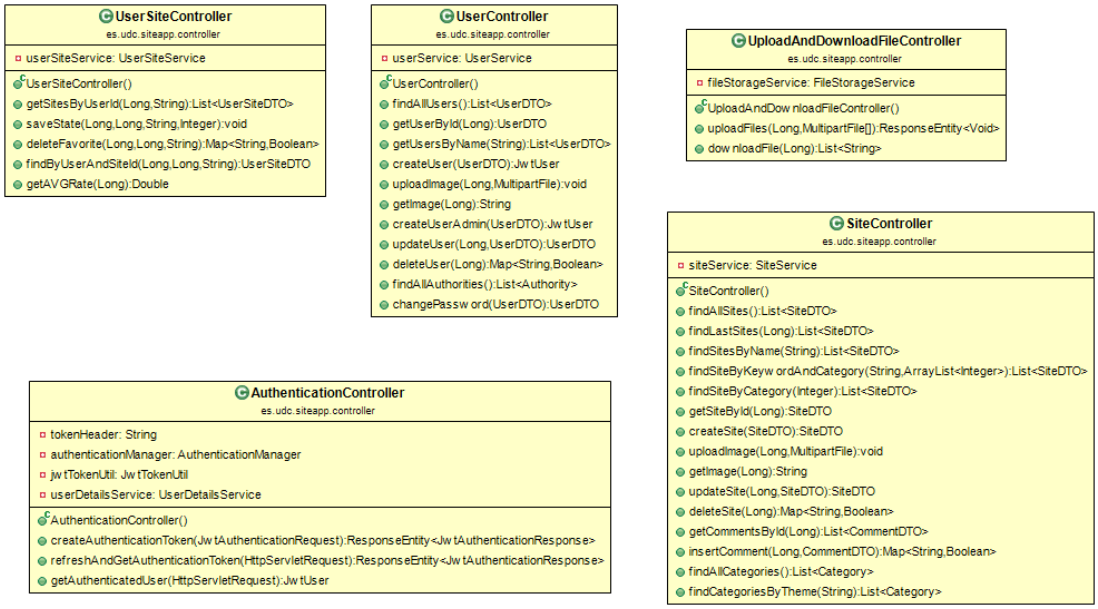

Figura 6.8: Controladores de la aplicación

A continuación, se exponen las diferentes peticiones que se pueden hacer desde el lado del cliente, agrupadas según el tipo de controlador en el que se encuentran en el servidor. En las tablas 6.1 y 6.2, podemos ver las peticiones correspondientes a *UserController* y *SiteController*, respectivamente. Además, tenemos la tabla 6.3 con las operaciones referentes a la autenticación mientras que, en la tabla 6.4, tenemos las peticiones para las relaciones entre usuario y sitio. Por último, el controlador que maneja la subida y descarga de imágenes complementarias para un sitio de interés se muestra en la tabla 6.5.

Como se ha comentado anteriormente, estos controladores harán uso de los servicios implementados en el servidor. En las tablas 6.6, 6.7, 6.8, 6.9, 6.10 se muestran las relaciones entre las peticiones habilitadas en los controladores y los correspondientes servicios que se llaman.

Operación	URL	Descripción
GET	/api/users	para devolver la lista con todos los usuarios creados
GET	/api/users/{id}	para obtener la información de un usuario concreto
POST	/api/users/admin	para crear un usuario con rol administrador
GET	/api/users/authorities	para obtener todos los roles disponibles para un usuario
PUT	/api/users/changePassword	para cambiar la contraseña de un usuario cuando la ha olvidado
POST	/api/users	para crear un usuario
DELETE	/api/users/delete/{id}	para eliminar un usuario concreto de la aplicación
GET	/api/users/image/{id}	para obtener la imagen de un usuario de la aplicación
GET	/api/users/name/	para obtener los datos de un usuario de la aplicación mediante su nombre
PUT	/api/users/update/{id}	para actualizar los datos de un usuario concreto
PUT	/api/users/upload/{id}	para actualizar la imagen de un usuario concreto

Cuadro 6.1: *Endpoints* correspondientes al *UserController*

Operación	URL	Descripción
GET	/api/sites	para devolver la lista con todos los sitios creados
GET	/api/sites/{id}	para obtener la información de un sitio concreto
GET	/api/sites/categories	para obtener todas las categorías posibles
GET	/api/sites/categories/theme	para obtener todas las categorías de un tema
POST	/api/sites/comment/{id}	inserta un comentario para un determinado sitio
GET	/api/sites/comments/{id}	obtiene los comentarios de un sitio
POST	/api/sites	para crear un sitio nuevo
DELETE	/api/sites/delete/{id}	para eliminar un sitio de interés en concreto
GET	/api/sites/filter	para realizar una búsqueda por diversos filtros
GET	/api/sites/filter/category	para devolver todos los sitios de una categoría
GET	/api/sites/image/{id}	para devolver la imagen principal de un sitio
GET	/api/sites/last	para devolver los últimos sitios introducidos en la aplicación
GET	/api/sites/name	para obtener los sitios que concuerdan con el nombre enviado
PUT	/api/sites/update/{id}	para actualizar un sitio de interés
PUT	/api/sites/upload/{id}	para actualizar la imagen principal de un determinado sitio

Cuadro 6.2: *Endpoints* correspondientes al *SiteController*

Operación	URL	Descripción
POST	/api/auth	para iniciar sesión de un usuario con su nombre de usuario y contraseña
GET	/api/info	para obtener la información del usuario autenticado en la aplicación
GET	/api/refresh	para refrescar el *token* del usuario conectado

Cuadro 6.3: *Endpoints* correspondientes al *AuthenticationController*

Operación	URL	Descripción
DELETE	/api/userSite/delete	para eliminar un sitio de la lista de favoritos o de la lista de pendientes
GET	/api/userSite/findByUserAndSiteId	para obtener una relación pasándole el usuario, el sitio y el tipo de relación
GET	/api/userSite/getAVGRate	para obtener la puntuación media de un sitio de interés
GET	/api/userSite/getSitesByUserId	para obtener la relación de un usuario y sitio de un tipo concreto
POST	/api/userSite/saveState	para guardar una relación usuario-sitio

Cuadro 6.4: *Endpoints* correspondientes al *UserSiteController*

Operación	URL	Descripción
GET	/api/downloads	para descargar las imágenes complementarias de un sitio de interés
POST	/api/uploads/{id}	para enviar al servidor las imágenes complementarias de un sitio de interés

Cuadro 6.5: *Endpoints* en el *UploadAndDownloadFileController*

Operación	URL	Método
GET	/api/users	findAll
GET	/api/users/{id}	findUserById
POST	/api/users/admin	registerUser
GET	/api/users/authorities	findAllAuthorities
PUT	/api/users/changePassword	changePassword
POST	/api/users	registerUser
DELETE	/api/users/delete/{id}	deleteById
GET	/api/users/image/{id}	getImage
GET	/api/users/name	getUsersByName
PUT	/api/users/update/{id}	updateUser
PUT	/api/users/upload/{id}	uploadImage

Cuadro 6.6: Relación servicio-controlador en el *UserController*

Operación	URL	Método
GET	/api/sites	findAll
GET	/api/sites/{id}	getSiteById
GET	/api/sites/categories	findAllCategories
GET	/api/sites/categories/theme	findCategoriesByTheme
POST	/api/sites/comment/{id}	insertComment
GET	/api/sites/comments/{id}	getCommentsById
POST	/api/sites	registerSite
DELETE	/api/sites/delete/{id}	deleteSiteById
GET	/api/sites/filter	findSiteByKeywordAndCategory
GET	/api/sites/filter/category	findSiteByCategory
GET	/api/sites/image/{id}	getImage
GET	/api/sites/last	findLastSites
GET	/api/sites/name	findSitesByName
PUT	/api/sites/update/{id}	updateSite
PUT	/api/sites/upload/{id}	uploadImage

Cuadro 6.7: Relación servicio-controlador en el *SiteController*

Operación	URL	Método
POST	/api/auth	generateToken
GET	/api/info	getUsernameFromToken
GET	/api/refresh	refreshToken

Cuadro 6.8: Relación servicio-controlador en el *AuthenticationController*

Operación	URL	Método
DELETE	/api/userSite/delete	deleteElement
GET	/api/userSite/findByUserAndSiteId	findByUserAndSiteId
GET	/api/userSite/getAVGRate	getAVGRate
GET	/api/userSite/getSitesByUserId	getSitesByUserId
POST	/api/userSite/saveState	saveState

Cuadro 6.9: Relación servicio-controlador en el *UserSitecontroller*

Operación	URL	Método
GET	/api/downloads	downloadFiles
POST	/api/uploads/{id}	storeFile

Cuadro 6.10: Relación servicio-controlador en el *UploadAndDownloadFileController*

6.1.3 Cliente

Como ya se ha indicado, la parte del cliente de la aplicación se ha desarrollado utilizando Vue, que es un *framework* open source de JavaScript, el cual nos permite construir interfaces de usuarios de una forma muy elegante mediante componentes. Un componente en Vue, en términos simples, es un elemento que permite encapsular código reutilizable en diferentes partes del *frontend*. Dentro de un componente, podremos encontrar tanto etiquetas HTML como estilos de CSS y/o código JavaScript. De esta forma, los componentes nos permiten desarrollar proyectos modularizados y fáciles de escalar, pudiendo reemplazar un componente de manera sencilla si así lo deseamos.
Una de las principales características es que se dispone de un DOM virtual que es capaz de reconocer las modificaciones en la página y actualizar solamente las partes que han sido cambiadas. Esto mejora la eficiencia y fluidez de la aplicación ya que no se tendrá que recargar totalmente una página para mostrar los cambios.
Para la **comunicación con el servidor** se realizarán peticiones HTTP a cada uno de los controladores con los métodos REST disponibles en el servidor. Estas peticiones se harán a través de **axios**, como se pueden ver en la figura 6.9, que es una librería de JavaScript construida con el objetivo de gestionar la programación asíncrona con promesas.

```
loadCategories() {
  axios
    .get("/api/sites/categories")
    .then((response) => {
      this.categories = response.data;
    });
},
```

Figura 6.9: Ejemplo de petición con axios

Componentes del cliente

En esta sección se mostrarán los diferentes componentes que se han diseñado para el cliente, realizando una breve explicación de para qué se utiliza cada uno de ellos. En la tabla 6.11 tenemos los componentes para el usuario, en la tabla 6.12 tenemos los componentes para sitios y en la tabla 6.13 tenemos los componentes comunes de la aplicación.

Componentes para usuario	Descripción del componente
Register	Componente utilizado para el registro e inicio de sesión de un usuario.
UserProfile	Componente que muestra la información del usuario conectado y ofrece la posibilidad de editar los datos del perfil del usuario. Además, se visualizarán las listas de sitios favoritos y de sitios pendientes de visitar.
AdminPanel	Componente que muestra el panel del administrador en el que puede gestionar los usuarios y los sitios de interés.

Cuadro 6.11: Componentes del usuario

Componentes para sitios	Descripción del componente
NewSiteForm	Componente utilizado para la creación de un sitio de interés.
EditSiteForm	Componente utilizado para la edición de un sitio de interés.
SiteDetails	Componente que muestra toda la información para un sitio de interés.
LastSites	Componente para visualizar los últimos sitios introducidos en la aplicación.
Seeker	Componente para realizar una búsqueda específica en la aplicación.
Categories	Componente para encontrar sitios filtrando por diferentes categorías.

Cuadro 6.12: Componentes de los sitios de interés

Componentes comunes	Descripción del componente
MenuBar	Visualiza la barra de navegación de toda la aplicación, con las diferentes opciones disponibles.
Home	Página principal de la aplicación con las distintas opciones de navegación.
Map	Página con el mapa interactivo de la aplicación.
Notification	Componente que se encarga de mostrar alertas y mensajes en la aplicación.
ErrorPage	Componente que se muestra cuando una acción no funciona de manera correcta.

Cuadro 6.13: Componentes comunes

6.1.4 Seguridad

Uno de los requisitos no funcionales, descritos en el capítulo de análisis, era que la aplicación debía asegurar la privacidad de los datos y que se debe disponer de un sistema de autenticación que distinga entre usuarios anónimos, usuarios registrados y usuarios administradores. Para la identificación de los usuarios, se ha optado por el estándar JWT, el cual asigna un *token* único a cada usuario que inicia sesión de forma correcta en la aplicación, indicando los permisos que posee. Gracias a esto, el servidor no se verá obligado a guardar el estado de cada usuario, reduciendo el número de consultas a base de datos de manera considerable.
Además de esto, en la parte del cliente, cuando un usuario inicie sesión se guardará el *token* en el objeto *Storage* (API de almacenamiento web), que nos permite almacenar datos de manera local en el navegador y sin necesidad de realizar conexión alguna a una base de datos. Este *token* será luego enviado en la cabecera de las peticiones del cliente cuando sea necesario.

Capítulo 7

Implementación y pruebas

En este capítulo, se mostrarán los detalles más relevante con respecto a la implementación propiamente de la aplicación. Además de esto, se indicarán los tipos de pruebas que se han realizado para comprobar el correcto funcionamiento de la aplicación desarrollada.

7.1 Estructura de la aplicación

En primer lugar, se detalla la organización del proyecto con su correspondiente estructura de directorios tanto para el servidor como para el cliente, siguiendo el diseño explicado en el capítulo anterior.

7.1.1 Servidor

Como se puede apreciar en la figura 7.1, el servidor se compone de los siguientes módulos:

- **configuration:** Contiene las clases de configuración de la aplicación. Por un lado, *SecurityConfig* contiene la configuración de seguridad de la aplicación y cómo se deben de hacer las peticiones a la aplicación, mientras que *SwaggerConfig* contiene la configuración para utilizar SwaggerUI y mostrar una documentación detallada de la API REST. (figura 7.2).

- **controller:** Contiene los controladores de la aplicación que se utilizan para la comunicación con el cliente. Estos controladores son de tipo REST (figura 7.3).

- **exception:** Contiene las clases para manejar las diferentes excepciones que pueden surgir durante la ejecución de la aplicación (figura 7.4).

- **model:** Contiene las clases que implementan la lógica de negocio de la aplicación (figura 7.5).

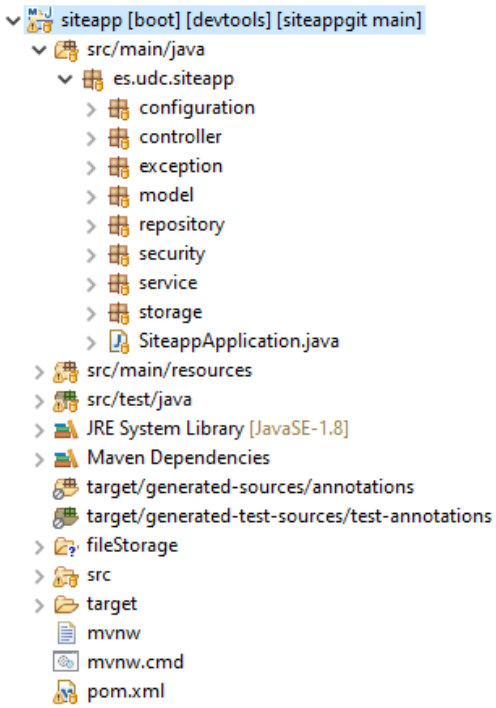

Figura 7.1: Estructura del servidor

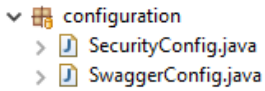

Figura 7.2: Detalle del directorio *configuration*

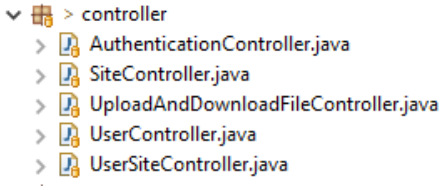

Figura 7.3: Detalle del directorio *controller*

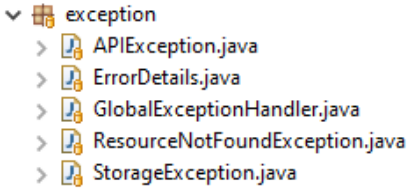

Figura 7.4: Detalle del directorio *exception*

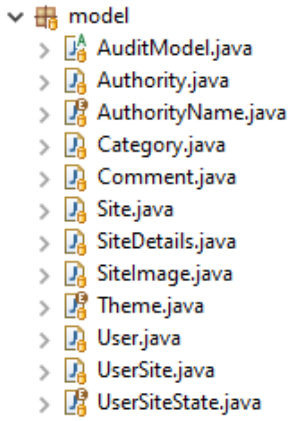

Figura 7.5: Detalle del directorio *model*

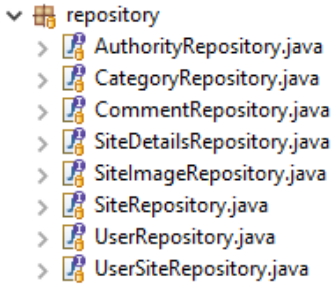

Figura 7.6: Detalle del directorio *repository*

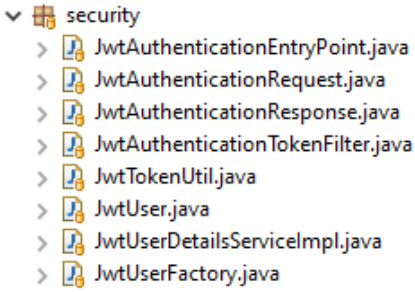

Figura 7.7: Detalle del directorio *security*

- **repository:** Contienen las interfaces con todas las conexiones que se necesita hacer a la base de datos (figura 7.6).

- **security:** Contiene las clases encargadas de la generación de *tokens* de acceso basados en las propias credenciales del usuario. De esa forma, la aplicación puede solicitar y enviar los *tokens* en las comunicaciones con el servidor (figura 7.7).

- **service:** Contiene las clases que implementan las operaciones necesarias en la aplica-

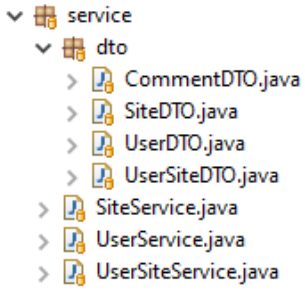

Figura 7.8: Detalle del directorio *service*

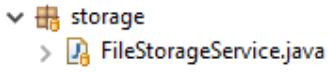

Figura 7.9: Detalle del directorio *storage*

ción. Además de esto, el directorio *dto* contiene los objetos que se intercambian entre servidor y cliente (figura 7.8).

- **storage:** Contiene la clase encargada de realizar la subida y descarga de las imágenes complementarias de los sitios de interés turístico. El directorio en el que se almacenan se configura en las propiedades de la aplicación (figura 7.9).

Además, tenemos los recursos necesarios para configurar la aplicación en el fichero ubicado en *src/main/resources/application.yml*. En este fichero podemos encontrar:

- Configuración del puerto en donde se va a lanzar la aplicación.
- Configuración del *driver* de postgreSQL para la conexión con base de datos.
- Configuración que específica el tamaño máximo de ficheros que se pueden intercambiar entre el cliente y servidor.
- Configuración de JWT para mantener la seguridad en la aplicación.
- Configuración del directorio para almacenar las imágenes complementarias de los sitios culturales.

Por último, se ha creado un *script* con los recursos necesarios para disponer de unos datos iniciales en la base de datos. Este *script* se encuentra en *src/main/resources/import.sql*.

7.1.2 Cliente

En la figura 7.10 se puede apreciar la estructura de módulos para el cliente. Cada componente de la lista que aparece en la figura cubre la funcionalidad correspondiente descrita con anterioridad en la subsección de componentes del cliente (sección 6.1.3).

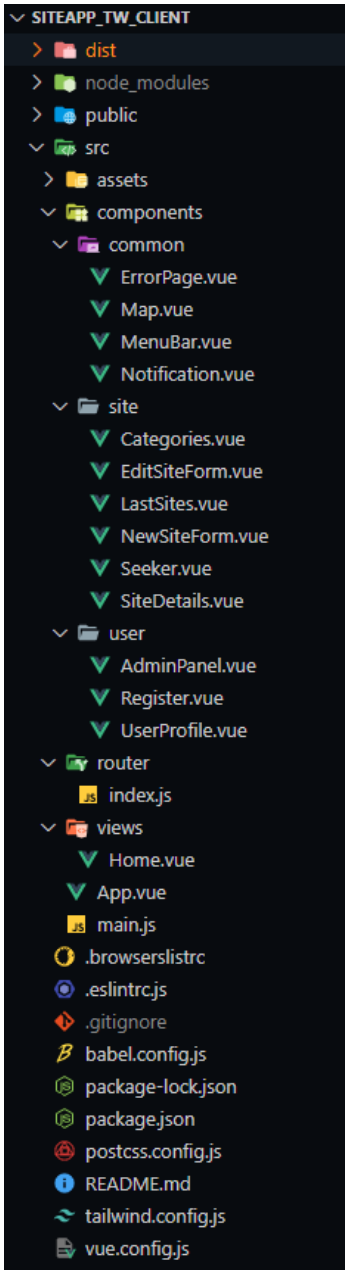

Figura 7.10: Estructura del cliente

7.2 Pruebas

En esta sección se recoge la información relativa a los distintos tipos de validaciones que se han realizado a lo largo del proyecto. Como se indica en la metodología de **Scrum**, al finalizar cada *sprint* es necesario que se realicen un conjunto de pruebas que deben ser superadas para validar el correcto funcionamiento de las funcionalidades implementadas durante ese *sprint*.
Las pruebas son una parte fundamental del desarrollo software. Éstas evalúan la fiabilidad de un producto y constituyen una excelente manera de comprobar que se cumple con los objetivos marcados. A través de las pruebas se pueden identificar errores en la implementación, calidad o funcionalidad de una aplicación y así subsanarlos. En este caso, para probar la aplicación que se ha desarrollado, se han realizado varios tipos de pruebas: pruebas de unidad, pruebas de integración, pruebas de sistema y pruebas de aceptación.

- **Pruebas unitarias:** Sirven para validar el correcto funcionamiento de cada una de la funcionalidades implementadas. Estas pruebas han sido llevadas a cabo sobre los métodos de los diferentes módulos del sistema que compone el servidor. Estas pruebas son el primer filtro para localizar los errores porque son ejecutadas sobre un elemento concreto. Estas pruebas se han realizado de manera sistemática utilizando JUnit, que es un *framework* que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.

- **Pruebas de integración:** Validan el correcto funcionamiento de la aplicación para un conjunto de funcionalidades entre las que existe cierta interacción. Las pruebas de integración deben contemplar todas las operaciones de la aplicación y todas las posibles acciones. Se han realizado pruebas sobre los siguientes casos:

  - Interacción entre el módulo *repository* y la base de datos.
  - Interacción entre el módulo *service* y *repository*.
  - Interacción entre el módulo *controller* y *service* utilizando Postman.
  - Interacción entre el cliente web y el API REST expuesto por el servidor.

- **Pruebas de sistema:** Validan el correcto funcionamiento de la aplicación como un sistema. Para la realización de estas pruebas, se testearon de manera manual todas las posibles interacciones de los diferentes tipos de usuario con la aplicación comprobando que todo funciona según lo establecido y que los resultados obtenidos se correspondían con lo esperado.

- **Pruebas de aceptación:** Tienen la finalidad de comprobar que se cumplen con las especificaciones marcadas en el análisis. Entre otras cosas, se ha comprobado que la aplicación fuese segura, que se diese la posibilidad al usuario a escribir la contraseña en claro si lo necesitase pero que guardase la contraseña cifrada en base de datos. También se ha comprobado que la aplicación tuviese un diseño web adaptable, para que no funcionase incorrectamente en los diferentes tamaños de dispositivos que existen en la actualidad: portátiles, móviles, *tablets*, etc. Otro punto muy importante es que el mapa interactivo, cuando tuviese una gran cantidad de puntos, funcionase con fluidez debido a que en las aplicaciones actuales al llenarse de puntos el mapa y tener muy poco filtrado, navegar por el mapa se vuelve una tarea ardua y tediosa.
  En este caso, la aplicación web ha sido analizada por los directores del proyecto que han actuado como *Product Owner* a lo largo del desarrollo. El resultado de estas pruebas ha sido satisfactorio en general y el cliente ha determinado que los diferentes incrementos de la aplicación iban cumpliendo con lo que se había planteado al principio de cada *sprint*. De igual forma, la valoración final de la aplicación ha sido muy positiva.

7.3 Problemas encontrados

En esta sección se explicarán muy brevemente los principales problemas encontrados en el desarrollo de la aplicación:

1. **Problemas en la petición POST con imágenes con axios:** Cuando se envía una imagen con axios mediante el método POST, no se pueden enviar más parámetros de otro tipo en la petición. Por tanto, ha sido necesario implementar un método para enviar las imágenes al servidor que contuviese el identificador del sitio de interés en el *endpoint* al que se llamaba.

2. **Problemas con nombres de entidades con palabras reservadas:** Al realizar la implementación en inglés, surgieron varios problemas con el nombre de algunos campos; por ejemplo, la entidad *user* se guarda en base de datos con el nombre users debido a que user es una palabra reservada en PostgreSQL. Lo mismo ha sucedido con el atributo *constraint* de la entidad *Comment* de la aplicación.

3. **Problemas relacionados con el mapa:** Cuando se estaba desarrollando la parte funcional del mapa interactivo surgieron varios problemas.

   - A la hora de importar Leaflet en una aplicación desarrollada con Vue.js, el marcador azul que se muestra por defecto para ubicar cada punto, no se carga correctamente. Después de investigar por internet, se ha descubierto que para ello es

necesario que en el *main.js* del proyecto se elimine y después se inserte el marcador y su sombra a mano.

- También a la hora de introducir otro marcador que no fuese el que aparece por defecto, en el caso del proyecto el marcador que indica el aparcamiento de un sitio, tampoco cargaba la imagen correctamente, ya que es necesario cargarlo en el componente concreto utilizando la directiva *require()* con el *path* de la imagen como parámetro.

4. **Problemas de control de acceso HTTP (CORS):** Al tener que lanzar el servidor en un puerto y el cliente en otro a la hora de desarrollar apareció el problema de intercambio de recursos de origen cruzado. Para subsanar este problema, hubo que crear un fichero en el cliente con el nombre de *vue.config.js* en el que se le introducía la dirección IP en donde se alojaba el servidor. De este modo se solventó el problema al realizar peticiones a la API REST.

Capítulo 8

Solución desarrollada

En este capítulo, vamos a mostrar las principales características de la aplicación que se ha desarrollado. El objetivo es hacer una pequeña guía ilustrativa con las funcionalidades más representativas de la aplicación.

8.1 Página principal

Un usuario que no se ha registrado podrá acceder de manera limitada a las funcionalidades de la aplicación. Como se puede ver en la figura 8.1, la aplicación web cuenta con una página principal en la que se mostrarán las diversas funcionalidades principales de la aplicación: ir al buscador, ir a los últimos sitios culturales introducidos en la aplicación, ir al mapa interactivo o ir al filtrado por categorías de la aplicación.

8.2 Acceso a la aplicación

Para acceder a la aplicación como usuario registrado, previamente se debería haber creado una cuenta. Para ello existe una página, accesible desde la página principal, que unifica las funcionalidades de darse de alta o de acceder a la aplicación como se puede ver en la figura 8.2. Una vez se ha creado la cuenta, se podrá acceder con posterioridad introduciendo las credenciales de acceso. Con motivo de facilitarle al usuario el acceso, en el campo de la contraseña hay un botón con un ojo que al pulsarlo muestra la contraseña en claro que se está introduciendo. Además de esto, y con el mismo motivo que lo anterior, se ha integrado la funcionalidad de recuperación de contraseña.

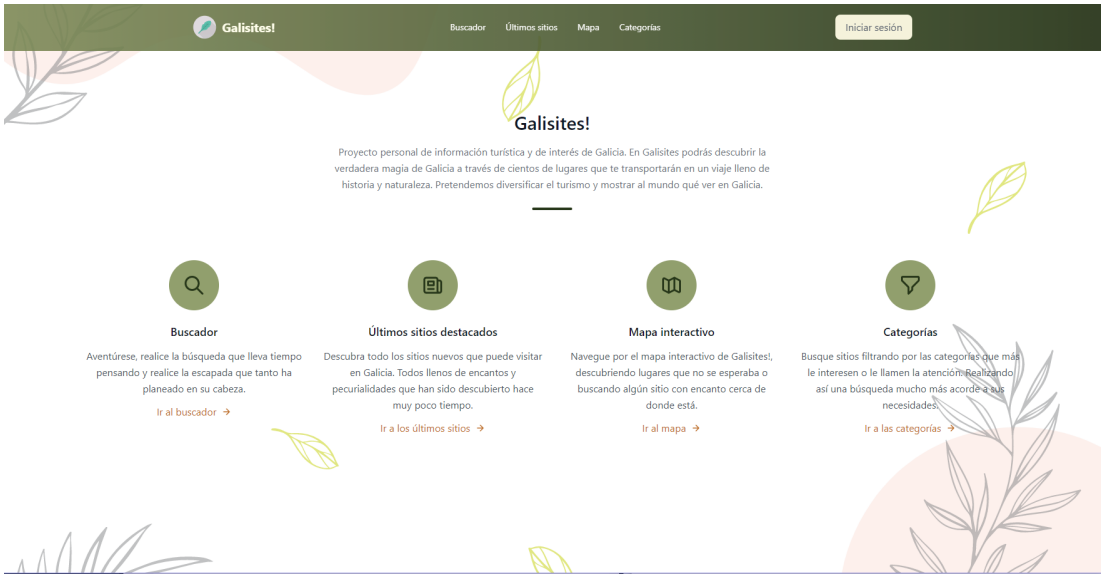

Figura 8.1: Página principal de la aplicación

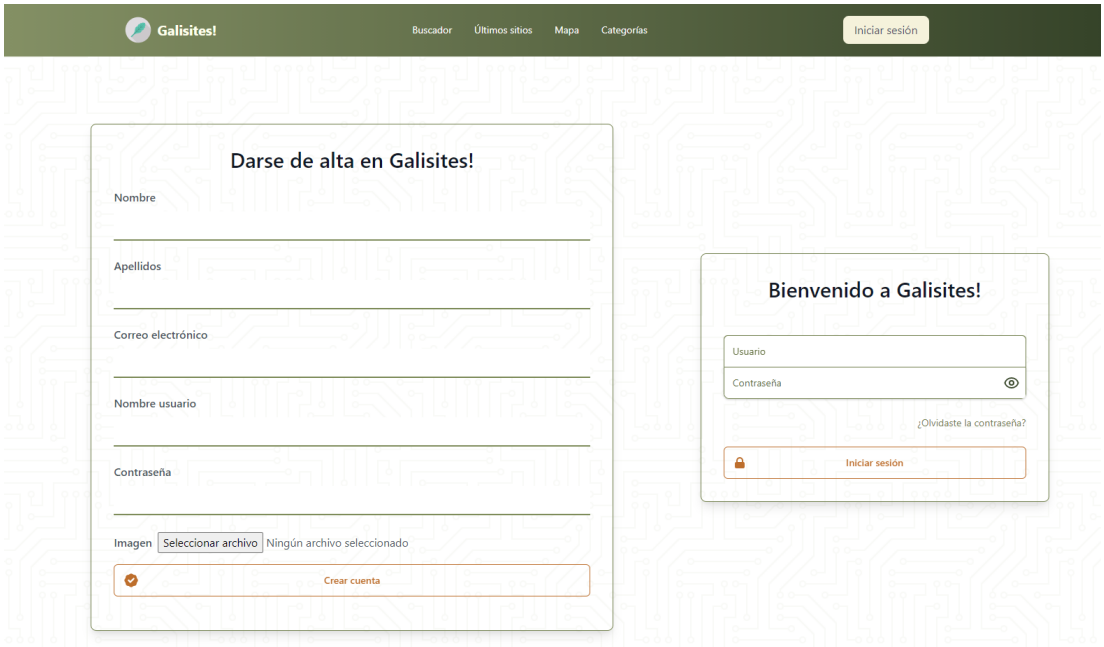

Figura 8.2: Página de acceso/registro de la aplicación

8.3 Búsqueda de un sitio

Al ser una aplicación de sitios culturales o naturales turísticos de Galicia, cuenta con un buscador que integra los diferentes filtros con las categorías que existen en la aplicación. Como se puede ver en la figura 8.3, al realizar una búsqueda se mostrarían los resultados ordenados

alfabéticamente y con una paginación simple para hacer más sencilla la facilidad de uso de la aplicación. Además de esto, filtrar los sitios en los que se puede ir con niños o se puede ir en automóvil.

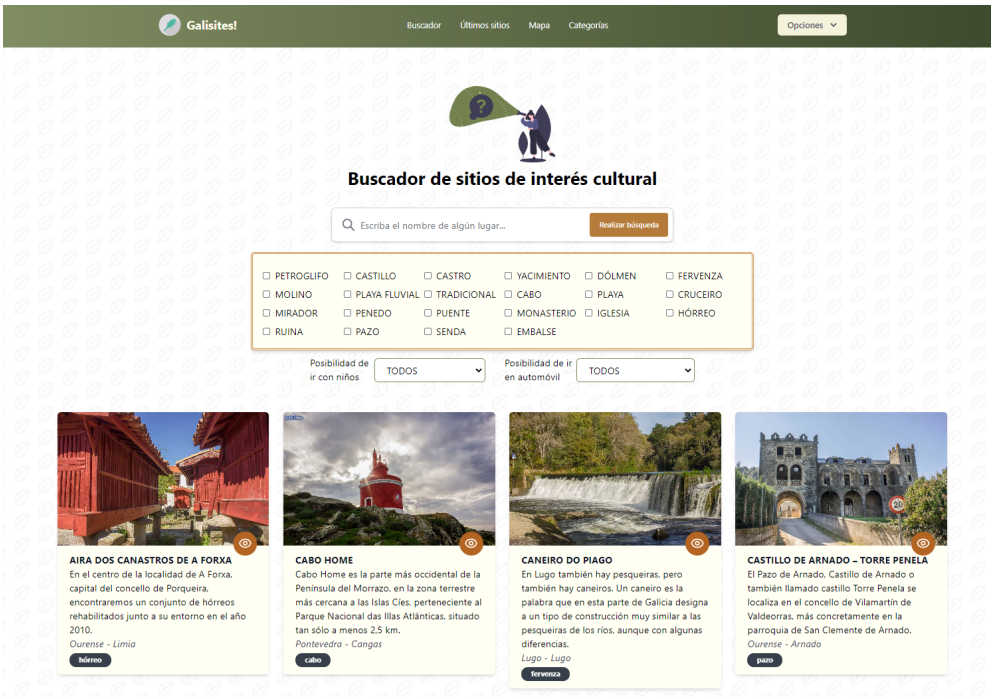

Figura 8.3: Buscador de la aplicación

8.4 Últimos sitios introducidos en la aplicación

La aplicación cuenta con una página en dónde se pueden ver los últimos veinte sitios introducidos en la aplicación. En esta página (figura. 8.4) se puede seleccionar una categoría y se mostrarían los últimos veinte sitios de esa categoría que se han introducido en la aplicación, ordenados por la fecha más reciente y con una paginación simple para no hacerle tedioso el uso al usuario.

8.5 Mapa interactivo

Al tratarse de una aplicación con ubicaciones geográficas, esta cuenta con un mapa interactivo en el que se puede navegar e ir descubriendo todos los sitios culturales y turísticos de la comunidad gallega. Si el usuario está registrado, al entrar en el mapa se cargarán las ubicaciones de los sitios que tiene en su lista de favoritos, mientras que si el usuario está navegando

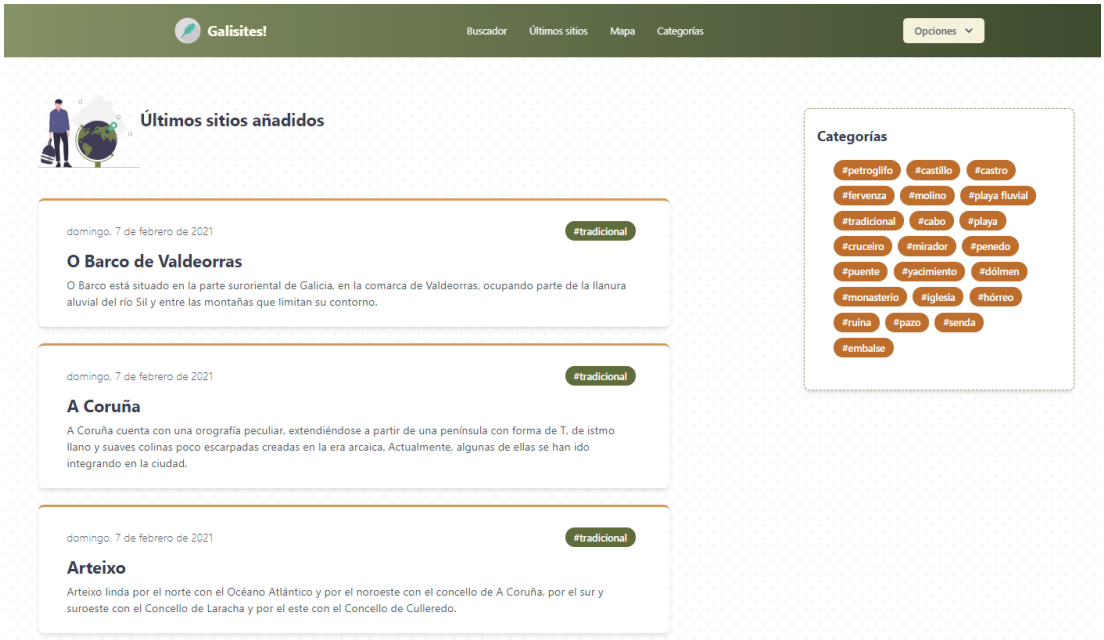

Figura 8.4: Página de últimos sitios de la aplicación

de forma anónima se cargarán los últimos veinte sitios introducidos por el administrador en la aplicación.

Como se puede ver en la figura 8.5 y para añadirle valor al mapa interactivo, están disponibles diferentes capas de vista del mapa: una vista normal que todo usuario esperaría al navegar en un mapa en la web, una vista satélite para poder ayudar a que el usuario conozca el terreno que ha descubierto, una vista callejera en el que se especifica el nombre de todas las calles y por último, pero no menos importante, una vista turística en la que se muestran los diferentes barrios en el mapa. Todo ello para adaptarse a las necesidades de los usuarios dependiendo de lo que busquen.

Además de esto, el usuario tiene a su disposición diferentes filtros para que pueda realizar de manera rápida una búsqueda más específica. Los sitios de interés que resulten de la búsqueda se muestran en el mapa en función de sus coordenadas.

Para que el mapa funcione correctamente aunque haya un número elevado de puntos sobre el mismo, a medida que se va cambiando el *zoom*, los puntos se agrupan o desagrupan para mejorar la visibilidad en el mapa. Cuando existe un grupo de puntos, si se hace clic sobre él estos despliegan los marcadores sobre el mapa pudiendo hacer clic sobre cada uno. Cuando un usuario hace clic en un marcador, se le muestra la información detallada del sitio, y si el sitio que ha seleccionado tiene ubicación de aparcamiento, se mostraría también un marcador de donde se encuentra.

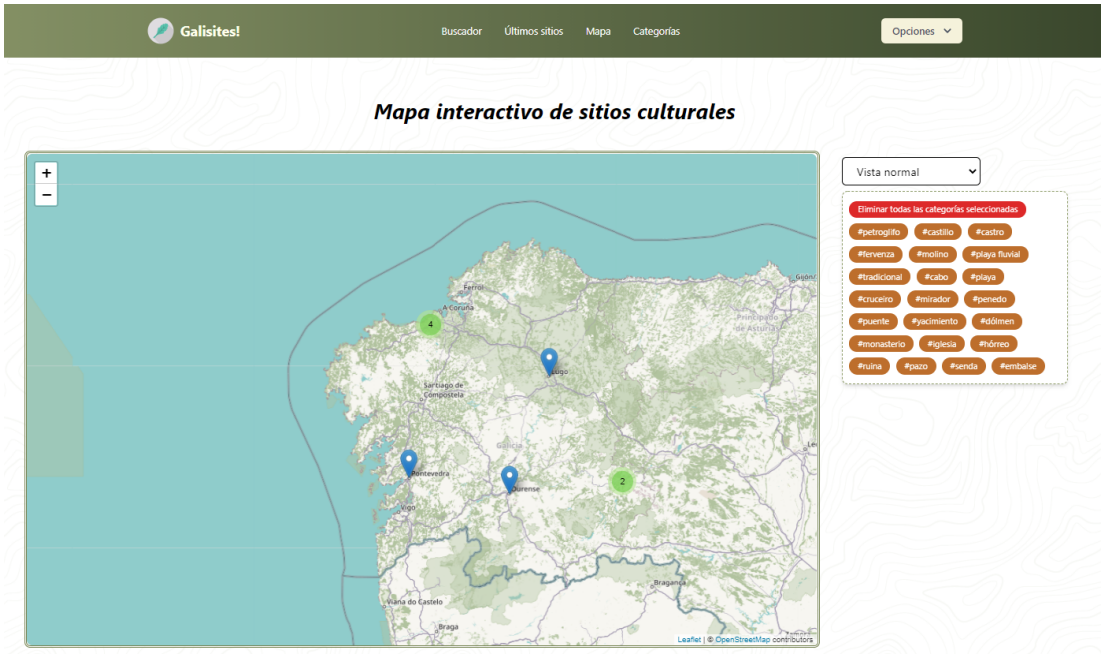

Figura 8.5: Mapa interactivo de la aplicación

8.6 Perfil del usuario

En el perfil del usuario, como podemos ver en la figura 8.6, es en donde podremos encontrar la lista con los sitios favoritos y los sitios pendientes de visitar. Además, desde este punto se pueden editar los datos del usuario o eliminarlo si así lo desea.

8.7 Funcionalidades de administración

Si el usuario es un usuario con rol de administrador, tendrá a su disposición un panel para gestionar los usuarios y los sitios de interés tal y como podemos ver en la figura 8.7. En él podrá buscar un usuario concreto o un sitio en particular y eliminarlo. En el caso de los sitios de interés, además de eso tendrá una opción para editarlos si fuese necesario.
El usuario administrador también cuenta con un formulario para dar de alta un nuevo sitio de interés como se ve en la figura 8.8. En este formulario se indicarán los campos que son obligatorios para la creación de un sitio nuevo.

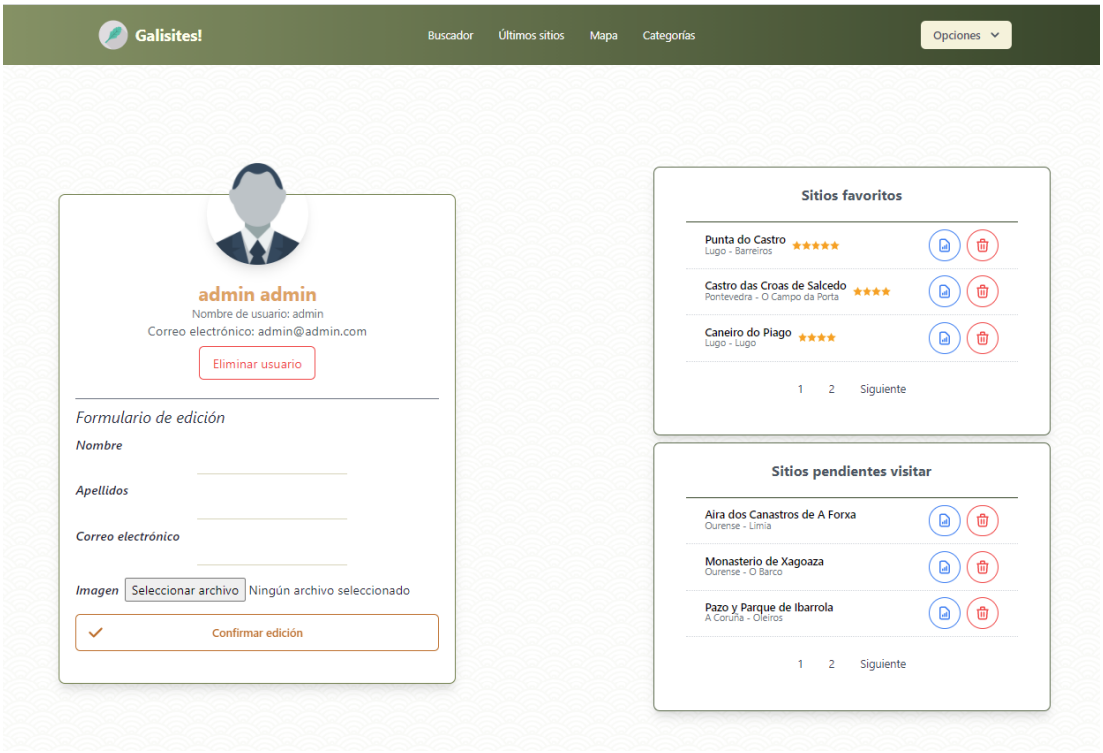

Figura 8.6: Perfil de un usuario de la aplicación

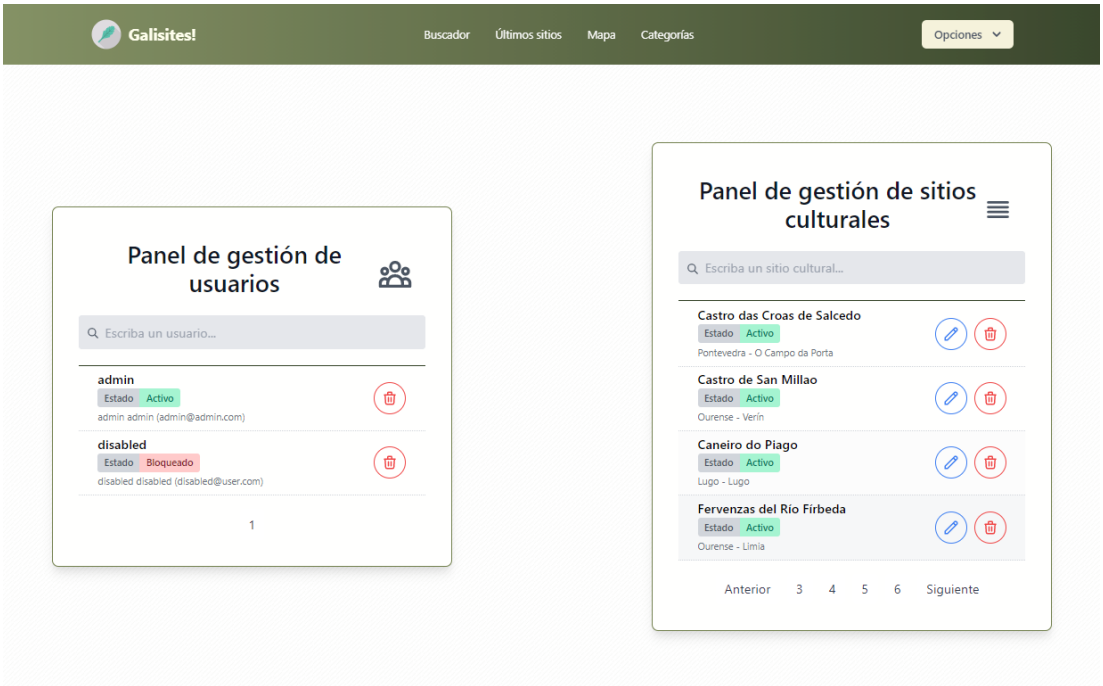

Figura 8.7: Panel de administrador de la aplicación

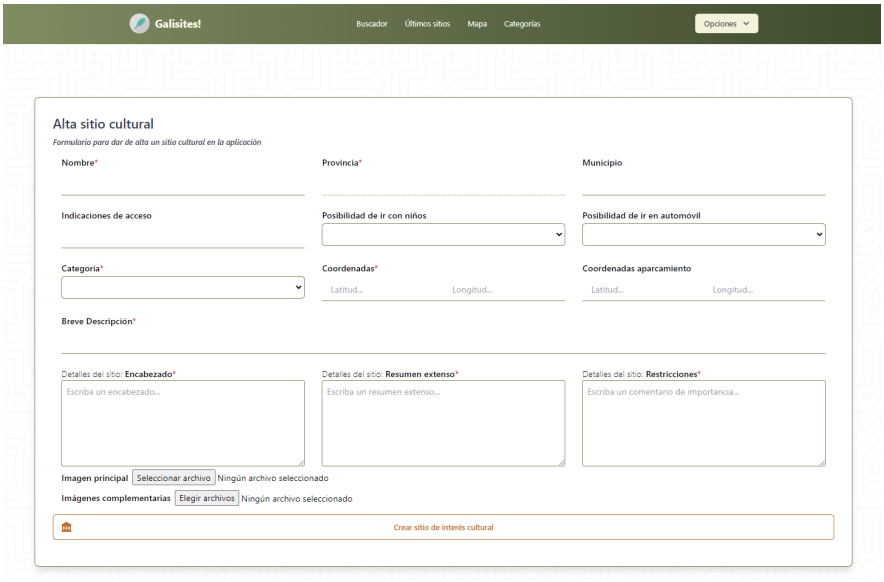

Figura 8.8: Formulario de alta de sitio cultural de la aplicación

8.8 Detalles de un sitio cultural

En este apartado se puede destacar el cuadro informativo que resume toda la información principal del sitio. Contribuye mucho a la usabilidad porque proporciona mucha información con un rápido vistazo. La aplicación está dotada de múltiples formas de encontrar sitios culturales o históricos de Galicia. Siempre que se realice una búsqueda se va a mostrar una información básica sobre el sitio, si se quiere una información más detallada hay que acceder a la información detallada del sitio en cuestión. Como se ve en la figura 8.9, en esta página se encuentra toda la información almacenada del sitio en cuestión, con sus imágenes complementarias, su cuadro con información importante, su valoración, sus funcionalidades para guardar el sitio en las diferentes listas de los usuarios y su tablón de comentarios de los usuarios. Un usuario anónimo no puede puntuar ni comentar un sitio, al igual que no puede almacenarlo en ninguna lista.

8.9 Categorías

En la figura 8.10 podemos ver la página de filtrado de los temas y las categorías de la aplicación. En ella se podrán encontrar los sitios de manera sencilla seleccionando el tema y la categoría que quiera el usuario y mostrándole sitios que cumplan esas condiciones. Los usuarios podrán acceder al sitio que deseen para ver la información detallada del mismo.

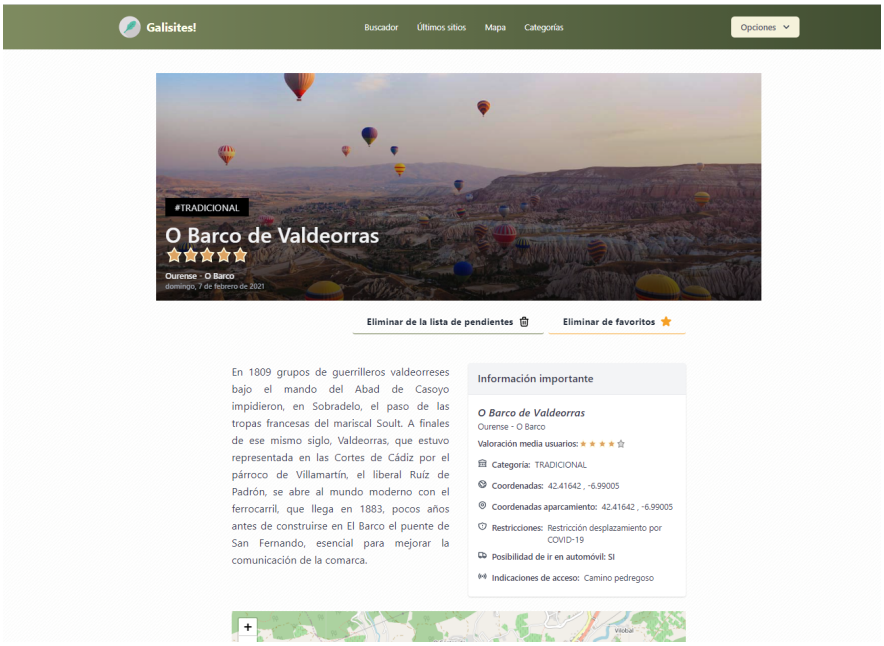

Figura 8.9: Página de detalles de un sitio en la aplicación

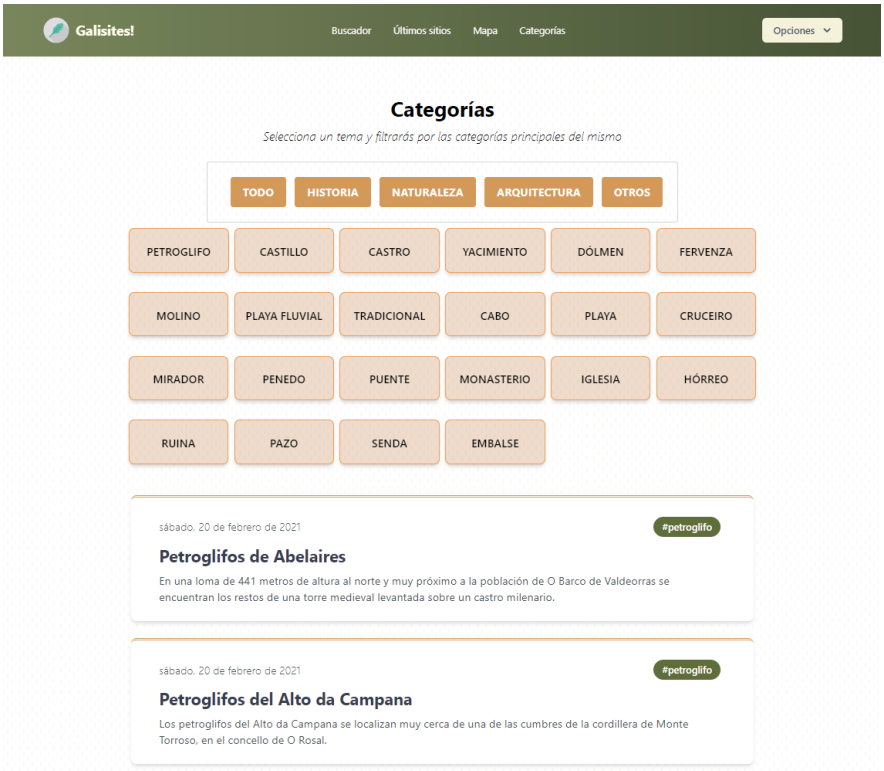

Figura 8.10: Página con las categorías en la aplicación

Capítulo 9

Conclusiones y trabajo futuro

9.1 Conclusiones

Una vez que se ha finalizado el desarrollo del proyecto, se ha hecho una recapitulación de los objetivos propuestos al principio y a lo largo del proceso de desarrollo de la aplicación, y se ha obtenido una serie de conclusiones:

- Al realizar un proyecto completo desde cero de una aplicación web, se han adquirido conocimientos que van desde el análisis previo y estudio de la tecnología, pasando por la planificación, seguimiento, documentación, análisis, diseño, implementación, pruebas y puesta en producción.
- Al poder tener más libertad que durante el grado para elegir las herramientas y tecnologías a utilizar se han desarrollado diferentes capacidades para afrontar problemas y buscar la solución de ellos. Además, al no estar fijadas las herramientas de antemano, proporciona un plus de motivación por aprender cosas nuevas que no se han visto durante el grado.
- Se ha desarrollado la aplicación cumpliendo los objetivos presentados al inicio de este documento, siguiendo una metodología personalizada basada en Scrum. Con esto, la aplicación cumple con los objetivos definidos y permite descubrir nuevos sitios culturales o históricos de Galicia.
- También he desarrollado un mayor manejo y comprensión en tecnologías próximas al entorno empresarial, adquiriendo experiencia en el manejo de Spring, Hibernate, Hibernate Spatial, Leaflet, PostGIS, Vuejs, Tailwind y la creación tanto de un servidor como de un cliente con sus comunicaciones vía REST.
- El resultado final de este desarrollo es un producto funcional que constituye una muy buena base para desarrollos futuros.

- Como conclusión personal, este proyecto ha supuesto un gran esfuerzo debido a la no dedicación completa a los estudios por motivos laborales. Todo este recorrido me ha servido para aprender una gran cantidad de nuevas tecnologías y conceptos que durante el grado no he visto y que se pueden poner en práctica a nivel laboral.

9.2 Líneas de trabajo futuro

Teniendo en cuenta el carácter didáctico y las restricciones temporales asociados a un trabajo de este tipo, es sencillo plantear posibilidades de mejoras. Algunas de ellas se mencionan a continuación:

- Se podría cambiar el sistema de almacenamiento de las imágenes por un sistema de repositorios que utilice una API ubicada en la **nube** como, por ejemplo, *Firebase* [56].
- Conexión con alguna web especializada en el **pronóstico del tiempo**. Sería interesante añadir este dato de manera directa en la información detallada de los sitios con el objetivo de ayudar al usuario a planificar su viaje.
- **Internacionalización** del sitio web para alcanzar una comunidad de usuarios mayor.
- Mostrar en cada lugar los puntos de interés por cercanía teniendo en cuenta un rango de kilómetros de **distancia**.
- Filtrar los sitios por otras características como tipo de acceso, proximidad del aparcamiento, etc.

Apéndices

Lista de acrónimos

AJAX Asynchronous JavaScript And XML. 18

API Application programming interface. 13

CORS Cross-Origin Resource Sharing. 70

CSS Cascading Style Sheets. 15

DI Dependency injection. 14

DOM Document Object Model. 17

HQL Hibernate Query Language. 49

HTML HyperText Markup Language. 15

IDE Integrated development environment. 13

IoC Inversion of Control. 14

Java EE Java standard edition. 15

Java SE Java enterprise edition. 15

JDBC Java Database Connectivity. 15

JPA Java persistence API. 15

JTS Java Topology Suite. 15

JWT JSON web token. 16

MVC Model-view-controller. 40

MVVM Model-view-viewmodel. 41

OGP Open Geospatial Consortium. 20

ORM Object-relational mapping. 15

REST Representational state transfer. 40

SQL Structured Query Language. 15

URI Uniform resource identifier. 42

Bibliografía

[1] (2021) Imagen metodología Scrum. [En línea]. Disponible en: https://www.pngwing.com/es/free-png-deqlm

[2] (2019) Impactur Galicia 2017. [En línea]. Disponible en: https://www.exceltur.org/wp-content/uploads/2019/01/IMPACTUR-Galicia-2017.pdf

[3] (2020) Galicia registra en el 2019 los mejores datos turísticos de su historia. [En línea]. Disponible en: https://www.turismo.gal/espazo-profesional/actualidade/detalle-nova?langId=es_ES&content=nova_2073.html

[4] (2021) Web de Galicia máxica. [En línea]. Disponible en: https://www.galiciamaxica.eu

[5] (2021) Web de Tripadvisor. [En línea]. Disponible en: https://www.tripadvisor.es

[6] (2021) Web de Minube. [En línea]. Disponible en: https://www.minube.com

[7] (2021) Portal oficial de turismo España. [En línea]. Disponible en: https://www.spain.info/es/

[8] (2021) Portal oficial de turismo Galicia. [En línea]. Disponible en: https://www.turismo.gal/inicio/

[9] (2021) Web de sitiosturisticos. [En línea]. Disponible en: https://sitiosturisticos.com

[10] (2021) Web de pgAdmin. [En línea]. Disponible en: https://www.pgadmin.org

[11] (2021) Web de PostgreSQL. [En línea]. Disponible en: https://www.postgresql.org

[12] (2021) Web de DBeaver community. [En línea]. Disponible en: https://dbeaver.io

[13] (2021) Web de Sourcetree. [En línea]. Disponible en: https://www.sourcetreeapp.com

[14] (2021) Web de Postman. [En línea]. Disponible en: https://www.postman.com

[15] (2021) Web de Spring Tools Suite. [En línea]. Disponible en: https://spring.io/tools

[16] (2021) Web de Visual Studio Code. [En línea]. Disponible en: https://code.visualstudio.com

[17] (2021) Web de Java. [En línea]. Disponible en: https://www.java.com/es/

[18] (2021) Web de Adopt Open JDK. [En línea]. Disponible en: https://adoptopenjdk.net

[19] (2021) Web de Oracle Java. [En línea]. Disponible en: https://www.oracle.com/es/java/

[20] (2021) Web de Spring. [En línea]. Disponible en: https://spring.io

[21] (2021) Web de Spring Boot. [En línea]. Disponible en: https://spring.io/projects/spring-boot

[22] (2021) Web de Hibernate. [En línea]. Disponible en: https://hibernate.org/orm/

[23] (2021) Wikipedia SQL. [En línea]. Disponible en: https://es.wikipedia.org/wiki/SQL

[24] (2021) Web de Hibernate Spatial. [En línea]. Disponible en: https://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html#spatial

[25] (2021) Introduction to Hibernate Spatial. [En línea]. Disponible en: https://www.baeldung.com/hibernate-spatial

[26] (2021) Web de Java Topology Suite. [En línea]. Disponible en: https://locationtech.github.io/jts/jts-features.html

[27] (2021) Web de Vuejs. [En línea]. Disponible en: https://vuejs.org/index.html

[28] (2021) Web de Tailwind. [En línea]. Disponible en: https://tailwindcss.com

[29] (2021) Web de Bootstrap. [En línea]. Disponible en: https://getbootstrap.com

[30] (2021) Web de Leaflet. [En línea]. Disponible en: https://leafletjs.com

[31] (2021) Github de Vue2Leaflet. [En línea]. Disponible en: https://github.com/vue-leaflet/Vue2Leaflet

[32] (2021) Web de JWT. [En línea]. Disponible en: https://jwt.io

[33] (2021) Github de axios. [En línea]. Disponible en: https://github.com/axios/axios

[34] (2018) Librería Axios: cliente HTTP para Javascript. [En línea]. Disponible en: https://desarrolloweb.com/articulos/axios-ajax-cliente-http-javascript.html

[35] (2021) Promises, MDN web docs. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Promise

[36] (2020) Introducción a las Promesas Javascript ES6. [En línea]. Disponible en: https://desarrolloweb.com/articulos/introduccion-promesas-es6.html

[37] (2021) Web de Javascript. [En línea]. Disponible en: https://www.javascript.com

[38] (2021) Wikipedia Javascript. [En línea]. Disponible en: https://es.wikipedia.org/wiki/JavaScript

[39] (2021) Wikipedia AJAX. [En línea]. Disponible en: https://es.wikipedia.org/wiki/AJAX

[40] (2021) Web de Swagger UI. [En línea]. Disponible en: https://swagger.io/tools/swagger-ui/

[41] (2021) El significado de base de datos relacional. [En línea]. Disponible en: https://www.oracle.com/es/database/what-is-a-relational-database/

[42] (2021) ¿Qué es una Base de Datos NoSQL? [En línea]. Disponible en: https://blogs.oracle.com/spain/qu-es-una-base-de-datos-nosql

[43] (2021) Wikipedia bases de datos espaciales. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Base_de_datos_espacial

[44] (2021) Wikipedia Sistema de información geográfica. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Sistema_de_información_geográfica

[45] (2021) PostGIS. [En línea]. Disponible en: https://postgis.net

[46] (2021) Opera, navegador web. [En línea]. Disponible en: https://www.opera.com/es

[47] (2021) Web de moqups. [En línea]. Disponible en: https://moqups.com

[48] (2021) Web draw.io online. [En línea]. Disponible en: https://app.diagrams.net

[49] (2021) Notion. [En línea]. Disponible en: https://www.notion.so/product

[50] (2021) Qué es Scrum? [En línea]. Disponible en: https://proyectosagiles.org/que-es-scrum/

[51] (2021) Wikipedia de Scrum. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)

[52] (2019) «BOE» núm. 251, de 18 de octubre de 2019, páginas 114772 a 114801. [En línea]. Disponible en: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2019-14977

[53] (2021) Wikipedia, MVC. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Modelo–vista–controlador

[54] (2021) El patrón MVVM. [En línea]. Disponible en: https://theshallowbay.github.io/tutoriales/2017/07/02/patron-mvvm-introduccion/#el-modelo

[55] (2021) Web project Lombok. [En línea]. Disponible en: https://projectlombok.org

[56] (2021) Firebase, Google. [En línea]. Disponible en: https://firebase.google.com/?hl=es-419