Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

Diseño e Implementación de una Aplicación Web de censos de aves urbanas para SEO/birdlife

Estudiante: Jessica Morado Rey
Dirección: José Losada Pérez

A Coruña, setembro de 2019.

A toda mi familia y amigos

Agradecimientos

Quiero agradecer primeramente a Juan Carlos del Moral por concederme la realización de este proyecto y por su enorme paciencia. A toda mi familia por su enorme apoyo, y en especial a mi pareja por haber compartido conmigo esta experiencia. También doy las gracias al director de este trabajo, José Losada Pérez, cuyos consejos y guías han sido de gran ayuda para mí.
Gracias.

Resumen

El objetivo de este proyecto es la implementación de una aplicación que permitirá a voluntarios de todo el país participar en censos de aves urbanas.

La aplicación estará orientada a trabajar con dos clases de especies de aves, aquellas denominadas coloniales, y las territoriales.

Cualquier usuario podrá registrarse en la aplicación y empezar a guardar información, identificando colonias o territorios, normalmente cerca de algún punto de paso habitual del usuario. Así se podrán realizar la mayor cantidad de visitas para que los datos sean lo más exahustivos posibles.

La metodología de desarrollo se basa en el Proceso Unificado de Desarrollo de Software. Esta metodología propone un trabajo iterativo e incremental que descompone la aplicación en pequeños proyectos y cada uno de estos subproyectos añade nuevas funcionalidades de forma progresiva.

Para el desarrollo de esta aplicación se utilizará el patrón Modelo-Vista-Controlador(MVC). Siendo el modelo basado en Symfony, un framework de PHP; y la vista-controlador basados en Angular, un framework de desarrollo basado en Javascript.

Abstract

The objective of this project is the implementation of an application that will allow volunteers from all over the country to participate in urban bird censuses.

The application will be oriented to work with two kinds of bird species, those called colonial birds, and territorial birds.

Any user will be able to register in the application and start saving information, identifying colonies or territories, usually near some habitual point of passage of the user. This way, the greatest number of visits can be made so that the data is as accurate as possible.

The development methodology is based on the Unified Software Development Process. This methodology proposes an iterative and incremental work that breaks down the application into small projects and each of these subprojects adds new functionalities progressively.

For the development of this application, the Model-View-Controller (MVC) pattern will be used. Being the model based on Symfony, a PHP framework; and the view-controller based on Angular, a development framework based on Javascript.

Palabras clave:

Symfony

PHP

Angular

Javascript

MVC

RUP

Censo aves urbanas

Censo SEO

Keywords:

Symfony

PHP

Angular

Javascript

MVC

RUP

Urban birds censuses

SEO census

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introdución

1.1 Motivación

El presente Trabajo de Fin de Grado consiste en el análisis, diseño e implementación de una aplicación web para realizar censos de aves urbanas.

Esta aplicación se realizará en colaboración con la Sociedad Española de Ornitología (SEO/-birdlife), por lo que la aplicación será gratuita y de código abierto, cediendo su uso a dicha sociedad.

El objetivo principal es reunir información procedente de voluntarios por toda España, para así conocer la población actual de estas aves (dónde crían y cuántos quedan), para trabajar en mantener sus efectivos y favorecer su conservación. La información generada por voluntarios, y su colaboración con expertos, contribuye a detectar asuntos que no habían sido advertidos y a identificar y llenar vacíos de información.

Las aves coloniales son todas aquellas especies que crían en grupo. Con 2 o más nidos/-parejas de una misma especie en un punto ya son considerados una colonia. Por el contrario, una especie territorial vive en solitario o en pareja en época de cría, sin establecer grupos de individuos. Suelen ser territoriales y procuran que no haya ejemplares de la misma especie criando en el mismo sitio.

Las especies de aves que forman colonias son menos abundantes y tienen áreas de distribución más amplias, además de ser más susceptibles de sufrir cambios en el tamaño de sus poblaciones, y por tanto, más vulnerables a la extinción. Las especies territoriales son a menudo especies sensibles y ya amenazadas, y el vivir en ciudades les añade un plus de peligrosidad. Es por todo esto que se desea controlar estas especies para conocer verdaderamente su estado de conservación.

1.2 Objetivos

Se quiere construir una nueva base de datos para guardar toda esta información, e integrar ésta con la base de datos de SEO, sobretodo para que no ocurran inconsistencias entre los usuarios de diferentes aplicaciones. Además esta integración también tiene por objetivo utilizar información ya predefinida, común a todas las aplicaciones de SEO y por lo tanto necesaria también en este proyecto.

Se crearán dos estructuras bien diferenciadas, una orientada a trabajar con los datos de aves coloniales y otra para tratar las aves territoriales.

Se necesitará un acceso restringido a la aplicación, ya que en ocasiones se genera cierta información sensible no disponible para todo el mundo. A coninuación se definen los siguientes objetivos principales:

- **Gestión de usuarios**. En términos generales, cada usuario podrá registrar información, consultar datos y obtener documentación informativa.
- **Gestión de roles**. La aplicación dispondrá de varios perfiles de usuario con funcionalidades diferenciadas. Se distingue entre usuario general y usuario administrador.
- **Gestión de colonias y territorios**. Se gestionarán los datos de colonias y territorios con características propias de cada estructura.
- **Identificación de colonias cercanas**. El usuario podrá visualizar en un mapa las colonias cercanas a él.
- **Consultas por parte de los usuarios**. Se ofrecerán consultas generales en base a filtros, y consultas más específicas por colonia o territorio.
- **Integración de bases de datos**. La aplicación debe poder registrar nuevos usuarios en la base de datos de SEO, así como consultar los ya existentes.

Capítulo 2

Fundamentos teóricos

2.1 Responsive Web

El diseño web responsive o adaptativo es una técnica de diseño web [1] que busca la correcta visualización de una misma página en distintos dispositivos. Desde ordenadores de escritorio a tablets y móviles. Se trata de redimensionar y colocar los elementos de la web de forma que se adapten a las dimensiones de cada dispositivo permitiendo una correcta visualización y una mejor experiencia de usuario. Como principales ventajas, destacan las siguientes:

Usabilidad. Si se usa una sola URL, a los usuarios les resulta más fácil compartir el contenido y enlazarlo.

Optimización. Requiere menos tiempo de ingeniería para mantener varias páginas para el mismo contenido. No es necesario realizar ningún redireccionamiento para que los usuarios lleguen a la vista optimizada para su dispositivo, de modo que se reduce el tiempo de carga.

Figura 2.1: Diferentes versiones en un dispositivo móvil

2.2 Modelo cliente servidor

Cuando se utiliza un servicio en Internet, como consultar una base de datos, transferir un fichero o participar en un foro de discusión, se establece un proceso en el que entran en juego dos partes [2] . Por un lado, el usuario, quien ejecuta una aplicación en el ordenador local: el denominado programa cliente. Este programa cliente se encarga de ponerse en contacto con el ordenador remoto para solicitar el servicio deseado. El ordenador remoto por su parte responderá a lo solicitado mediante un programa que está ejecutando. Este último se denomina programa servidor. Los términos cliente y servidor se utilizan tanto para referirse a los programas que cumplen estas funciones, como a los ordenadores donde son ejecutados esos programas.

2.3 Patrón MVC

Modelo Vista Controlador (MVC) [3] es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Se trata de un modelo muy maduro y que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo.

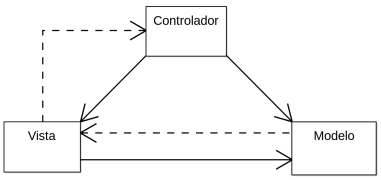

Figura 2.2: Patrón clásico MVC

El **Modelo** contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia. El modelo contiene el fin comercial de la aplicación [4]. Para aplicaciones simples de CRUD (Create, read, update, delete), éste suele ser un modelo de datos simple. Para aplicaciones más complejas, el modelo reflejará naturalmente ese aumento en la complejidad. En Angular, esta capa está codificada en los llamados servicios. El servicio será el encargado de ponerse en contacto con el modelo del servidor, el que realmente interactúa con la base de datos.

La **Vista**, o interfaz de usuario, compone la información que se envía al cliente y los mecanismos interacción con éste.

El **Controlador** actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

2.4 Lenguajes de servidor

Se conoce como *lenguaje del lado del servidor* [5] a aquel que se ejecuta en el servidor web, inmediatamente antes de que el sitio web se envíe a través de Internet al usuario. Los sitios web que se ejecutan en el servidor pueden realizar un amplio abanico de tareas hasta formar el propio sitio web que va a ver el usuario: acceso a base de datos, conexión en red... Los lenguajes más utilizados en la actualidad son: ASP, PERL y PHP.

2.5 Autorización y autenticación

La autenticación [6] se define como el proceso mediante el cual se verifica quién es el usuario, es decir, su ámbito se refiere a la identificación. La autorización es el proceso mediante el cual se verifica a qué se tiene acceso, es decir, su ámbito se limita al control de acceso.

En esta aplicación se utiliza el protocolo OAuth.v2 para el proceso de autorización. OAuth.v2 es un protocolo que se ha convertido en el estándar de facto en el uso de APIs. Permite a las aplicaciones un acceso limitado a los datos de los usuarios, sin tener que proporcionar las credenciales de dicho usuario, desacoplando la autenticación y la autorización a los datos.

Los *tokens* e identificadores que se manejan dentro de OAuth.v2 son:

- **Client-id**. Es un identificador único que representa la aplicación en el servidor de autenticación.
- **Client-secret**. Es una clave secreta que pertenece a la aplicación que es utilizada por el servidor de autorización para comprobar si dicha aplicación está autorizada.
- **Access-token**. Son claves proporcionadas por el servidor de autorización a la aplicación que permite el acceso a las APIs durante un tiempo limitado tras el cual el token deja de ser válido.
- **Refresh-token**. Es una clave también proporcionada por el servidor de autorización a la aplicación que permite solicitar nuevos access token después de que estos hayan caducado.

Para las tareas de Admnistración [7] se necesita autenticar a los usuarios, ya que por debajo se utiliza el sistema de seguridad de Symfony. La autenticación en Symfony puede parecer un poco ”mágica” al principio. Esto se debe a que, en lugar de crear una ruta y un

controlador para manejar el inicio de sesión, se activará un proveedor de autenticación: algún código que se ejecuta automáticamente antes de que se llame a su controlador.

Capítulo 3

Fundamentos tecnológicos

3.1 Capas Vista y Controlador

3.1.1 Javascript y Typescript

JavaScript [8] es un lenguaje de programación que permite realizar actividades complejas en una página web, como mostrar actualizaciones de contenido en el momento, interactuar con mapas, animaciones gráficas 2D/3D etc.

TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, que esencialmente añade tipos estáticos y objetos basados en clases. TypeScript extiende la sintaxis de JavaScript, por tanto cualquier código JavaScript existente debería funcionar sin problemas.

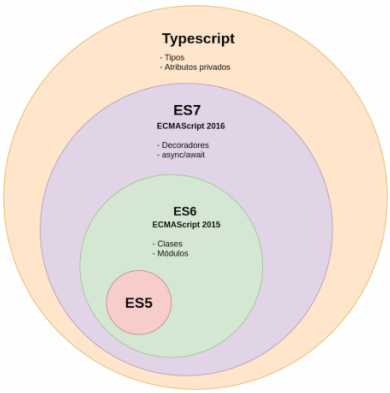

Figura 3.1: Conjunto de TypeScript

3.1.2 HTML y CSS

HTML [9], siglas en inglés de HyperText Markup Language (lenguaje de marcas de hipertexto), es un lenguaje de marcas para la elaboración de páginas web. Es un estándar ya que define una estructura básica y un código (denominado código HTML) para la definición del contenido de una página web.

CSS [10] (siglas en inglés de Cascading Style Sheets), en español «Hojas de estilo en cascada», es un lenguaje de diseño gráfico para definir y crear los estilos de una página web. Se utiliza conjuntamente con HTML para mejorar la apariencia de la interfaz gráfica.

3.1.3 Angular 6

Angular [11] es un framework de desarrollo para JavaScript creado por Google. La finalidad de Angular es facilitar el desarrollo de aplicaciones web SPA y además dar herramientas para trabajar con los elementos de una web de una manera más sencilla y optima.

Una aplicación web SPA creada con Angular es una web de una sola página, en la cual la navegación entre secciones y páginas de la aplicación, así como la carga de datos, se realiza de manera dinámica, casi instantánea y sobre todo sin refrescar la página en ningún momento.

3.2 Capa Modelo

3.2.1 PHP

PHP [12] (acrónimo recursivo de PHP: Hypertext Preprocessor) es un lenguaje de código abierto muy popular especialmente adecuado para el desarrollo web y que puede ser incorporado en páginas HTML.

Lo que distingue a PHP de lenguajes como Javascript es que el código es ejecutado en el servidor, generando HTML y enviándolo al cliente. El cliente recibirá el resultado de ejecutar el script, aunque no sabrá el código subyacente que era con anterioridad.

3.2.2 Symfony

[13] Symfony es un proyecto PHP de software libre que permite crear aplicaciones y sitios web rápidos y seguros de forma profesional.

Es un completo Framework [14] diseñado para optimizar, gracias a sus características, el desarrollo de las Aplicaciones web. Para empezar, separa la lógica de negocio, la lógica

de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación.

3.2.3 Twig

Twig es un motor de plantillas para utilizar con el lenguaje PHP. Convierte archivos predefinidos con distintas directrices en páginas HTML.

Capítulo 4

Estudio de viabilidad

Previo al comienzo de cualquier proyecto debe analizarse la viabilidad del mismo. El estudio de viabilidad dispone el éxito o fracaso de un proyecto a partir de una serie de datos. En este capítulo se lleva a cabo un análisis considerando tres aspectos fundamentales: la viabilidad económica, viabilidad técnica y viabilidad de mercado.

4.1 Viabilidad económica

La viabilidad ecónomica es el primer aspecto que se ha estudiado. Hay que tener en cuenta que este apartado engloba todos los gastos generados por los recursos humanos, los recursos *software* y los recursos *hardware*.

En cuanto a los recursos humanos, este proyecto lo ha desarrollado una sola persona con fines no lucrativos, por lo que el coste es mínimo.

En relación a los recursos *software*, cabe destacar que todo el desarrollo de la aplicación ha sido con *software* gratuito y de código abierto. Por otro lado, SEO utiliza *software* de pago de cara al cliente, principalmente el servidor web, que utiliza una licencia de tipo Windows. Aún así, es una licencia ya pagada y que no supondrá ningún coste extra, por lo que en este sentido se concluye que los recursos *software* tienen un coste mínimo.

En lo relativo a los recursos *hardware*, para el desarrollo sólo ha sido necesario un portátil con conexión a internet. SEO desplegará esta aplicación en un servidor ya configurado para otras aplicaciones, por lo que no se tendrá que contratar ningún nuevo tipo de servicio. Por lo tanto, los recursos *hardware* son fácilmente asimilables.

Por todos los motivos mencionados anteriormente, se puede decir que el proyecto es viable económicamente.

4.2 Viabilidad técnica

Las tecnologías utilizadas y explicadas en los capítulos 2 y 3, son ampliamente utilizadas y reconocidas en el desarrollo de *software*. Utilizando estas tecnologías se obtiene un producto fácilmente escalable en el tiempo y de fácil mantenimiento, por lo que el riesgo de utilizar las tecnologías escogidas y que el proyecto fracase, es bajo.

Por todo esto, se puede decir que el proyecto es viable técnicamente.

4.3 Viabilidad de mercado

SEO/birdlife utiliza el concepto de ”Ciencia ciudadana”, un término que se refiere a la investigación científica llevada a cabo por una suma de colaboradores, en su totalidad o en parte por científicos y profesionales junto a voluntarios sin formación profesional. Actualmente SEO/BirdLife mantiene 18 iniciativas de ciencia ciudadana, con diferente nivel de complejidad y de implicación, entre los que se incluyen los censos de aves.

Los censos específicos que promueve y organiza SEO/BirdLife cada año intentan conocer el tamaño de población y la distribución de una especie o grupo de especies concreto en cualquier época del año en España.

Con el fin de obtener esta información de la forma más ajustada posible, se elaboran unas instrucciones propias para cada caso que facilitan el trabajo a cientos de voluntarios. La metodología suele ser entregar una ficha para que la persona la imprima y vaya cubriendo datos en ella, junto con unas instrucciones. Más tarde los usuarios podrán informatizar estos datos, normalmente utilizando hojas de cálculo.

Entre las iniciativas de ciencia ciudadana que se mencionan antes, se encuentran los seguimientos de aves. Para estos seguimientos SEO/birdlife cuenta con una aplicación móvil disponible para Android e IOS, llamada ”Programas de Seguimiento”.

En el mercado actual no hay ninguna aplicación parecida para realizar el mismo cometido, y como esta aplicación formará parte del propio conjunto de aplicaciones de SEO, se concluye que el proyecto es viable en el mercado.

Capítulo 5

Metodologia de desarrollo

En este capítulo se dará una visión generalizada de la arquitectura del sistema, y se explicará la metodología empleada para el desarrollo del mismo.

5.1 Arquitectura del sistema

El sistema se ha diseñado siguiendo un patrón de diseño de modelo-vista-controlador (MVC) [15]. El patrón requiere que cada una de estas partes esté separada en distintos objetos.

El modelo contiene únicamente los datos puros de aplicación; no contiene lógica que describe cómo pueden presentarse los datos a un usuario.

La vista presenta al usuario los datos del modelo. La vista sabe cómo acceder a los datos del modelo, pero no sabe el significado de estos datos ni lo que el usuario puede hacer para manipularlos.

Por último, el controlador está entre la vista y el modelo. Escucha los sucesos desencadenados por la vista (u otro origen externo) y ejecuta la reacción apropiada a estos sucesos. En la mayoría de los casos, la reacción es llamar a un método del modelo.

Estas partes pueden subdivirse en más capas lógicas, como se observa en la figura 5.1 , además de poder implementarse con diferentes tecnologías.

La aplicación está compuesta por el "***front-end***", que engloba las capas vista y controlador y que está implementada con tecnologías muy diferentes a las del "***back-end***", compuesto por la capa modelo.

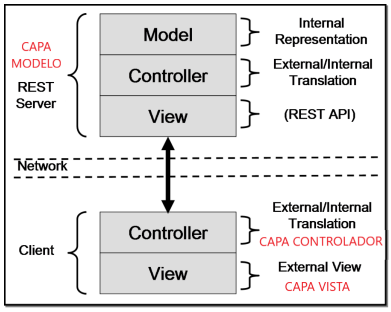

Figura 5.1: Diferentes capas del sistema

A continuación se hace una breve explicación de todas las capas lógicas en las que se ha dividido el sistema:

- **Capa acceso a datos**. Es la capa lógica que se encarga de acceder a la base de datos y recuperar, modificar o almacenar los datos solicitados por la capa de lógica de negocio. En esta capa se utiliza la tecnología de Doctrine, un mapeador objeto-Relacional que permite mapear objetos PHP a entidades en Base de datos y realizar operaciones sobre ellos a través de la implementación de repositorios y manejadores de entidades.

- **Capa lógica de negocio**. Es la capa en la que se implementa la lógica de los casos de usos especificados para el proyecto. Hace uso de diferentes clases de tipo *controlador*, que se encargan de procesar las solicitudes, aplicar la lógica de negocio y devolver una respuesta.

- **Capa servicios REST**. Esta capa se encarga de poner en contacto la capa controlador con la lógica del negocio. Se proporcionan diferentes APIs a las que el controlador puede realizar peticiones HTTP. Dependiendo de la solicitud ésta podrá ser resuelta directamente con la capa de acceso a datos y devolviendo una respuesta, o en caso de ser compleja delegando su procesado a la capa de lógica de negocio. Se ha implementando con el framework de ApiPlatform.

- **Capa controlador**. Esta capa provee diferentes servicios que pueden ser invocados desde la Vista. Los servicios tienen diferentes funciones en las que se recogen los datos provenientes de la vista, y se realizan las peticiones HTTP correspondientes a la capa Servicios Rest del modelo. Esta capa está implementada dentro del framework de Angular, haciendo uso del lenguaje Typescript, una extensión de Javascript.

- **Capa vista**. En esta capa se desarrolla la lógica de la interfaz de usuario. Interacciona con la capa controlador haciendo uso de sus servicios, ya sea proporcionándole datos o solicitándolos. Esta capa está implementada dentro del framework de Angular, haciendo uso de la tecnología HTML y CSS.

5.2 Metodología iterativa

La metodología que se usará en este proyecto se basará en el Proceso Unificado de Desarrollo de Software. Esta metodología propone un esquema de trabajo iterativo e incremental que descompone la aplicación en pequeños proyectos. Cada uno de estos subproyectos añade nuevas funcionalidades de forma progresiva. Las distintas iteraciones se centran en aspectos relevantes del software, empezando por aquellos que suponen un mayor riesgo para la consecución de los objetivos. En cada iteración se hace análisis, diseño, implementación y pruebas, de forma que cada iteración incorpora más funcionalidades sobre la anterior, hasta que en la última se termina con un software que implementa toda la funcionalidad. A continuación se enumeran todas las fases por las que debe pasar el proyecto siguiendo esta metodología:

- **Definición del proyecto**, Definición de la aplicación a desarrollar, en coordinación con el director del proyecto y con el responsable de SEO
- **Elaboración**. Establecimiento de requisitos y evaluación de los requerimientos y restricciones. Incluye el estudio y aprendizaje de las tecnologías a usar. Se definen las iteraciones de las que constará el proyecto,
- **Desarrollo**. Desarrollo de las distintas iteraciones. En cada una de ellas se realizará: análisis, diseño, implementación y pruebas.
- **Transición**. Fase que se basa en la prueba y entrega del producto al usuario final. Se realizarán diferentes pruebas para comprobar el correcto funcionamiento y la correcta solución de los requisitos establecidos.

5.3 Descripción de las iteraciones

- **Definición del proyecto**.En esta fase se ha llevado a cabo primeramente la definición del proyecto, en colaboración con el responsable de SEO/birdlife, Juan Carlos del Moral. Después de la definición se establecieron todos los requisitos que la aplicación debería cumplir, y a continuación se realizó un estudio de viabilidad.
- **Estudio de las tecnologías**. Se hizo un estudio sobre las tecnologías más adecuadas para la realización de este proyecto. En la fase de definición se establecieron diferentes preferencias sobre tecnologías ya conocidas, por lo que debía analizarse cuál utilizar.
- **Aprendizaje de las tecnologías**. Una vez definidas las tecnologías a utilizar en la aplicación, se ha realizado un toma de contacto con ellas para que el proceso de desarrollo resulte eficiente y correcto.

- **Puesta en marcha**. Una vez analizada toda la definición y requisitos del proyecto, se han formalizado todos los casos de uso. En esta fase también se ha creado el entorno necesario para el desarrollo de la aplicación, y una vez creado, se han generado los diferentes esqueletos que conformarán la estructura del proyecto.

- **Creación de entidades**. El desarrollo comienza en esta fase, donde se definen las entidades persistentes. Una vez definidas se procede a su implementación, que implica la creación de toda la base de datos. En este paso también se integra el framework de ApiPlatform, que autogenera APIs basadas en las entidades y que serán modificadas en iteraciones posteriores.

- **Seguridad**. En esta fase se añade el componente de seguridad a la aplicación, que permite securizar las APIs ya creadas y las que se creen más adelante. También permite securizar el acceso al propio sistema, codificar contraseñas y denegar accesos.

- **Creación de los controladores**. Se crean todas las clases que se van a encargar de procesar las peticiones de las Apis y aplicar la lógica de negocio. Se eliminarán aquellas APIs autogeneradas que no son necesarias, y se gestionarán aquellas que por su complejidad requieren de un controlador que aplique la lógica de negocio.

- **Creación de la Administración**. Se requiere implementar una vista para que los usuarios administradores puedan interactuar directamente con la base de datos. Se añade el componente de EasyAdmin, un framework que creará toda esta vista automáticamente. Es suficiente con indicar qué entidades se quieren administrar, e indicarle al componente de seguridad qué roles tendrán acceso a esta vista.

- **Creación de servicios**. Se crean una serie de servicios con diferentes funciones que encapsulan la lógica del negocio.

- **Creación de los componentes generales**. En la capa vista se crearán los componentes comunes a todas las páginas, tales como el login o el registro.

- **Creación de los componentes de Aves coloniales**. Se crean todos los componentes visuales para la estructura de Aves coloniales y se enlazan con los servicios ofrecidos por la capa controlador.

- **Creación de los componentes de Aves territoriales**. Se crean todos los componentes visuales para la estructura de Aves territoriales y se enlazan con los servicios ofrecidos por la capa controlador.

- **Internacionalización**. En iteraciones anteriores se siguió un diseño que permitiera añadir fácilmente otros idiomas para traducir la aplicación. Una ver terminados todos los componentes, se añaden hojas de texto estructuradas con las traducciones deseadas.
- **Creación de loa memoria**. Por último se han recopilado los pasos seguidos en el desarrollo e implementación de este proyecto, y se han escrito en esta memoria

Capítulo 6

Análisis de requisitos

En este capítulo se tratarán en detalle los requisitos para la aplicación. Se explicarán aquellos que afectan directamente al desarrollo del software, ya sea por requerimientos que la aplicación debe cumplir, o por necesidades específicas sobre el sistema usado.

6.1 Requisitos funcionales

En este apartado se describen en términos generales aquellos requisitos que la aplicación deberá satisfacer. Un requisito define una función del sistema de software o sus componentes. El funcionamiento detallado de cada requisito se especifica en la sección 6.4.

- **Manejar dos grandes estructuras de aves**. La aplicación debe ser útil para dos grandes grupos de especies coloniales y no coloniales. Tienen pantallas en común pero pantallas propias de cada grupo.
- **Autenticación y autorización**. Dos niveles de autorización: El usuario general que puede añadir datos y hacer consultas y el usuario administrador con muchas más funciones propias de administración.
- **Integración con BBDD de usuarios**. Podrán interactuar con el sistema aquellos usuarios ya existentes en la base de datos de usuarios de SEO, así como poder registrarse desde la aplicación en la misma base de datos.
- **Documentación**. Los usuarios podrán obtener archivos que el administrador ha subido previamente al servidor y que pueden resultar interesantes para ellos.
- **Registro de una colonia o territorio**. Se registrarán datos para su posterior análisis. Estas colonias y territorios deben poder ser validados por un administrador, así como poder identificar que el territorio o la colonia han quedado vacíos.

- **Añadir datos Colonia/Territorio**. Se debe poder guardar datos diferentes para cada año para la misma colonia o territorio, así como poder modificar ciertos datos que el usuario pudo introducir por error, o que cambiaron en un corto período de tiempo.
- **Completar datos municipio**. Se podrá indicar si los censos de las colonias asignadas a un municipio forman un censo completo o no. También se podrá indicar si el conjunto de todas las visitas de una colonia forman un censo completo de ésta misma.
- **Identificar colonias/territorios cercanas**. Se podrá visualizar la posición del usuario y las colonias cercanas. Con respecto a los territorios, el usuario podrá ver su rango de búsqueda y sólo el número de territorios dentro de ese rango.
- **Grabar coordenadas**. Se abrirá Google Maps cuando sea necesario y el usuario podrá seleccionar (marcando un punto) las coordenadas desde el mapa.
- **Consultas**. Se podrán hacer consultas generales y específicas, de ambas estructuras, y con diferentes filtros para cada una.

6.2 Actores

La aplicación reconoce a dos actores, el usuario general y el usuario administrador. El usuario general es un voluntario o voluntaria que interacciona con la aplicación de censos, ya sea introduciendo datos u obteniéndolos. El usuario administrador es aquella persona que deseablemente es ajena al equipo informático, encargada de realizar diferentes funciones de administración de la aplicación.

6.3 Requisitos no funcionales

- **Eficiencia** La aplicación manejará grandes cantidades de datos (habrá miles de colonias y territorios por toda España), por lo que las interacciones con la base de datos deberán ser lo más rápidas y eficientes posibles.
- **Usabilidad** Deberá crearse una interfaz gráfica fácil de usar e intuitiva. El contenido visual deberá ser lo más parecido posible al estilo de diseño de las aplicaciones web de SEO/birdlife. Tendrá que ser un diseño adaptable a pequeños y medianos dispositivos porque la mayoría de las veces los usuarios lo utilizarán desde un teléfono móvil.
- **Seguridad** La aplicación deberá garantizar la privacidad de los datos de los usuarios, así como utilizar un protocolo de intercambio de datos seguro. La mejor opción es utlizar HTTPS y usar un certificado ya instalado en el servidor de SEO/birdlife para garantizar la autenticidad de la página.

- **Escalabilidad** Este proyecto deberá ser fácilmente escalable para poder seguir desarrollándolo en un futuro. El catálogo de especies podrá aumentar o disminuir según las circunstancias, además de que a la aplicación se le irán añadiendo funcionalidades diferentes a lo largo del tiempo.
- **Internacionalización** Se ofrecerá la opción de escoger un idioma para mostrar el contenido visual. En función de los idiomas en los que SEO tiene guardados los nombres de las especies, se decidió traducir la aplicación a: Español, Gallego, Vasco, Catalán e Inglés.

6.4 Especificación de casos de uso

Es una técnica para la captura de requisitos potenciales de un nuevo sistema. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar la aplicación con el usuario o con otro sistema para conseguir un objetivo específico.

6.4.1 Casos de uso comunes a ambas estructuras

CU-101: Registrarse	
Descripción	Registro de un nuevo usuario en el sistema
Actores	Usuario normal
Precondiciones	Usuario todavía no registrado.
Flujo	1. El usuario avanza a Registrar 2. El sistema muestra el formulario estándar de SEO para el registro de usuarios. 3. El usuario rellena los campos obligatorios y si quiere, los opcionales, y completa el registro . 4. Se envía una petición con los datos al servidor de SEO y se registra el usuario en el sistema.

Tabla 6.1: CU-101 Registrarse

CU-102: Identificarse	
Descripción	Identificación de un usuario en el sistema
Actores	Usuario normal; Usuario administrador
Precondiciones	Usuario debe estar registrado en la base de datos de SEO
Flujo	1. El usuario rellena los campos con su nombre de usuario y contraseña y solicita su identificacón. 2. El sistema envía una una petición a SEO para comprobar su autenticidad. Si todos los datos son correctos, se devolverá la información del usuario. 3. El sistema recoje el email devuelto en el paso 2, y envía una petición de autorización a la aplicación. 4. Se recoge el código de autorización devuelto por la aplicación, junto con otros datos, y se guarda en memoria. 5. El sistema navega a la página de selección de especies

Tabla 6.2: CU-102: Identificarse

CU-103: Selección	
Descripción	Selección de Aves coloniales y no coloniales respectivamente.
Actores	Usuario normal; Usuario administrador
Precondiciones	Usuario identificado
Flujo	1. Si el usuario desea trabajar con todas las especies de Aves coloniales, seleccionará la opción de Aves coloniales. 1.1 Se navega al menú general de Aves coloniales, donde se muestra un listado de todas ellas. 2. Si el usuario desea trabajar con todas las especies de Aves No coloniales, seleccionará la opción de Aves no coloniales. 2.2 Se navega al menú general de Aves no coloniales, donde se muestra un listado de todas ellas.

Tabla 6.3: CU-103: Selección

CU-104: Desconexión	
Descripción	Desconexión de de la aplicación
Actores	Usuario normal; Usuario administrador
Precondiciones	Estar en menú de Aves coloniales o no coloniales.
Flujo	1. Usuario seleccionará su nombre en la barra superior de navegación. 2. Se despliega un submenú con su Identificador de usuario, su correo, y la opción de Desconectar. 3. El suario seleccionará la opción de Desconectar y el sistema limpiará toda la memoria junto con los datos guardados del usuario. Se realiza una petición de desconexión a la aplicación. 4. El sistema navega a la pagína inicial.

Tabla 6.4: CU-104: Desconexión

CU-105: Ver mis visitas	
Descripción	Ver las visitas registradas por el usuario
Actores	Usuario normal; Usuario administrador
Precondiciones	Haber pulsado Ver mis visitas en el menú lateral.
Flujo	1. El usuario accede a la página de Mis visitas, donde se muestran todas aquellas registradas por el usuario identificado. 2. El usuario puede utilizar diferentes filtros para afinar su búsqueda.. 3. El usuario puede elegir entre ver las visitas registradas en Territorios o en Colonias 4. Sólo Administrador. Puede descargar los datos obtenidos en formato excel.

Tabla 6.5: CU-105: Ver mis visitas

CU-106: Estadísticas generales de Aves coloniales	
Descripción	Ver las estadísticas generales de Aves coloniales
Actores	Usuario normal; Usuario administrador
Precondiciones	Haber pulsado Consultas generales->Aves coloniales en menú lateral.
Flujo	1. El usuario navegará a la página de estadísticas de Aves coloniales 2. El usuario puede utilizar diferentes filtros para afinar su búsqueda. 3. El usuario verá la información devuelta por el sistema. 4. Sólo Administrador. Puede descargar los datos obtenidos en formato excel.

Tabla 6.6: CU-106: Estadísticas generales de Aves coloniales

CU-107: Estadísticas generales de Aves no coloniales	
Descripción	Ver las estadísticas generales de Aves no coloniales
Actores	Usuario normal; Usuario administrador
Precondiciones	Haber pulsado Consultas generales->Aves no coloniales en menú lateral
Flujo	1. El usuario navegará a la página de estadísticas de Aves no coloniales 2. El usuario puede utilizar diferentes filtros para afinar su búsqueda. 3. El usuario verá la información devuelta por el sistema. 4. Sólo Administrador. Puede descargar los datos obtenidos en formato excel.

Tabla 6.7: CU-107: Estadísticas generales de Aves no coloniales

CU-108: Documentación	
Descripción	Obtener documentación alojada en el servidor web.
Actores	Usuario normal; Usuario administrador
Precondiciones	Haber pulsado Documentación en menú lateral
Flujo	1. El usuario navegará a la página de Documentación. 2. El usuario puede descargar la documentación que desee.

Tabla 6.8: CU-108: Documentación

6.4.2 Casos de uso de Aves coloniales

CU-201: Registrar Colonia	
Descripción	Registrar los datos de una colonia
Actores	Usuario normal; Usuario administrador
Precondiciones	Estar en menú de Aves coloniales.
Flujo	1. El usuario selecciona la especie con la que quiere trabajar. 2. El sistema la guarda en memoria y la imprime en la barra superior . 3. El usuario avanza a la página Registrar Colonia. 4. El usuario rellenará todos los datos obligatorios, pasando las diferentes páginas del formulario, y aquellos opcionales que considere oportunos. 5. Se mostrará un resumen con la información final de los datos que el usuario ha introducido. 6. Se persiste la información.

Tabla 6.9: CU-201: Registrar Colonia

CU-202: Ver colonias	
Descripción	Ver los datos de las colonias registradas
Actores	Usuario normal; Usuario administrador
Precondiciones	Estar en menú de Aves coloniales.
Flujo	1. El suario selecciona la especie con la que quiere trabajar. 2. El sistema la guardará en memoria y la imprimirá en la barra superior . 3. El usuario navegará a la página Ver colonias. 4. El usuario puede utilizar los filtros disponibles para afinar su búsqueda. 5. Una vez que el sistema ha recuperado los datos, se imprime una lista con las colonias que cumplen los filtros (si aplican). 6. El usuario puede visualizar los datos de cada colonia. 7. Sólo Administrador. Tendrá la opción de descargar el listado, y la información de cada colonia en formato excel.

Tabla 6.10: CU-202: Ver colonias

CU-203: Ver colonias cercanas	
Descripción	Ver los datos de las colonias registradas cercanas a la posición del usuario
Actores	Usuario normal; Usuario administrador
Precondiciones	Estar en menú de Aves coloniales.
Flujo	1. El usuario selecciona la especie con la que quiere trabajar. 2. El sistema la guardará en memoria y la imprimirá en la barra superior . 3. El usuario navegará a Colonias cercanas 4. El usuario puede seleccionar diferentes distancias disponibles para ampliar o reducir el radio de busqueda. 5. Una vez que el sistema ha recuperado los datos, se imprime en el mapa los puntos donde se encuentran las colonias. 6. El usuario puede visualizar los datos de cada colonia encontrada.

Tabla 6.11: CU-203: Ver colonias cercanas

CU-204: Estadísticas por especie	
Descripción	Ver estadísticas correspondientes a la especie seleccionada
Actores	Usuario normal; Usuario administrador
Precondiciones	Estar en menú de Aves coloniales.
Flujo	1. El usuario selecciona la especie con la que quiere trabajar. 2. El sistema la guardará en memoria y la imprimirá en la barra superior . 3. El suario navegará a Estadísticas 4. El usuario podrá utilizar diferentes filtros y apartados para afinar su búsqueda. 5. Sólo Usuario Administrador. En cada apartado podrá descargar los datos obtenidos en formato excel.

Tabla 6.12: CU-204: Estadísticas por especie colonial

CU-205: Estadísticas de nidos	
Descripción	Ver estadísticas de nidos de la especie seleccionada
Actores	Usuario normal; Usuario administrador
Precondiciones	Tener seleccionada una especie de ave colonial
Flujo	1. El usuario navegará a Estadísticas de nidos. 2. El usuario podrá utilizar diferentes filtros y apartados para afinar su búsqueda.

Tabla 6.13: CU-205: Estadísticas de nidos de especie colonial

CU-206: Completar censo de municipio	
Descripción	Completar los datos de censo de un municipio
Actores	Usuario normal; Usuario administrador
Precondiciones	Tener seleccionada una especie de ave colonial
Flujo	1. El usuario navegará a Censo municipio 2. El usuario deberá indicar el municipio y la temporada en la que quiere introducir datos. 3. El sistema devolverá los datos obtenidos. Si no hay datos el usuario podrá crear un nuevo censo. 4. El usuario podrá asignar y desasignar colonias al censo de municipio. 5. El usuario podrá indicar que el censo del municipio está completo o no.

Tabla 6.14: CU-206: Completar censo municipio

CU-207: Registrar visita en colonia	
Descripción	Registrar los datos de una nueva visita en una colonia
Actores	Usuario normal; Usuario administrador
Precondiciones	Haber pulsado Ver visitas en cualquier ficha de información de una Colonia.
Flujo	1. El usuario navegará hasta la página de visitas, donde se mostrarán todas aquellas registradas en la colonia seleccionada. 2. Se creará una nueva visita. 3. El usuario cubrirá los datos obligatorios, y si lo desea, los opcionales. 4. El usuario registra la visita y el sistema persistirá los datos. 6. Sólo Administrador. Podrá descargar los datos obtenidos en formato excel.

Tabla 6.15: CU-207: Registrar visita en colonia

6.4.3 Casos de uso de Aves no coloniales

CU-301: Registrar Territorio	
Descripción	Registrar los datos de un territorio
Actores	Usuario normal; Usuario administrador
Precondiciones	Estar en menú de Aves no coloniales.
Flujo	1. El usuario selecciona la especie con la que quiere trabajar. 2. El sistema la guardará en memoria y la imprimirá en la barra superior . 3. El usuario nevagará a la página Registrar Territorio. 4. El usuario rellenará todos los datos obligatorios y aquellos opcionales que considere oportunos. 5. El usuario registra el territorio y el sistema persistirá la información

Tabla 6.16: CU-301: Registrar Territorio

CU-302: Registrar visita en territorio	
Descripción	Registrar los datos de una nueva visita en un territorio
Actores	Usuario normal; Usuario administrador
Precondiciones	Haber pulsado Ver visitas en cualquier ficha de información de una Territorio.
Flujo	1. El usuario navegará hasta la página de visitas, donde se mostrarán todas aquellas registradas en el territorio seleccionado. 2. Se podrá registrar una nueva visita 3. El usuario cubrirá los datos obligatorios, y si lo desea, los opcionales. 4. Se registra la visita y el sistema persistirá los datos. 6. Sólo Administrador. Podrá descargar los datos obtenidos en formato excel.

Tabla 6.17: CU-302: Registrar visita en territorio

CU-303: Ver territorios	
Descripción	Ver los datos de los territorios registrados
Actores	Usuario normal; Usuario administrador
Precondiciones	Estar en menú de Aves no coloniales.
Flujo	1. El usuario selecciona la especie con la que quiere trabajar. 2. El sistema la guardará en memoria y la imprimirá en la barra superior . 3. Usuario navegará a la página Ver territorios. 4. El usuario podrá utilizar los filtros disponibles para afinar su búsqueda. 5. Una vez que el sistema ha recuperado los datos, se imprime una lista con los territorios que cumplen los filtros (si aplican). 6. El usuario puede visualizar los datos de cada territorio. 7. Sólo Administrador. Tendrá la opción de descargar el listado, y la información de cada territorio en formato excel.

Tabla 6.18: CU-303: Ver territorios

CU-304: Ver territorios cercanos	
Descripción	Ver número de territorios cercanos
Actores	Usuario normal; Usuario administrador
Precondiciones	Estar en menú de Aves no coloniales.
Flujo	1. El usuario selecciona la especie con la que quiere trabajar. 2. El sistema la guardará en memoria y la imprimirá en la barra superior . 3. El usuario navegará a Territorios cercanos 4. El usuario seleccionar diferentes distancias disponibles para ampliar o reducir el radiode busqueda. 5. Una vez que el sistema ha recuperado los datos, se imprime el número de territorios encontrados.

Tabla 6.19: CU-304: Ver territorios cercanos

CU-305: Estadísticas por especie no colonial	
Descripción	Ver estadísticas correspondientes a la especie
Actores	Usuario normal; Usuario administrador
Precondiciones	Estar en menú de Aves no coloniales.
Flujo	1. El usuario selecciona la especie con la que quiere trabajar. 2. El sistema la guardará en memoria y la imprimirá en la barra superior . 3. El usuario navegará a Estadísticas 4. El usuario podrá utilizar diferentes filtros y apartados para afinar su búsqueda. 5. Sólo Usuario Administrador. En cada apartado podrá descargar los datos obtenidos en formato excel.

Tabla 6.20: CU-305: Estadísticas por especie no colonial

Capítulo 7

Planificación y costes

Es importante para la gestión de un proyecto conocer dos factores relevantes como son el tiempo y el coste.

7.1 Planificación

Para una buena planificación hay que tener en cuenta los recursos de los que se disponen y que pueden ser de varios tipos. En este caso se consideran los recursos humanos y los recursos técnicos.

7.1.1 Recursos humanos

A pesar de que el proyecto ha sido desarrollado por una única persona, ésta ha adquirido diferentes perfiles en el proceso y los cuales se explican a continuación:

- **Gestor del proyecto**. Aquella persona que se encarga de gestionar un proyecto desde la fase inicial hasta la final, mediante un plan de proyecto.
- **Analista**. Es el encargado del desarrollo en lo que respecta a su diseño y asimilación de los requisitos, así como de analizar las posibles utilidades y modificaciones necesarias del sistema para una mayor eficacia.
- **Programador**. Es el encargado de implementar todo el código de la aplicación siguiendo las directrices del analista.

7.1.2 Recursos técnicos

Para el desarrollo del sistema se utilizó un ordenador portatil con conexión a Internet, y cuyas especificaciones se mencionan a continuación:

- **Marca y modelo**. Asus Strix
- **Procesador**. Intel(R) Core (TM) i7-7700HQ
- **Memoria RAM**. 8GB
- **Almacenamiento**. Disco SSD de 128GB

7.2 Costes

Como se mencionó anteriormente, la única persona desarrolladora de este proyecto lo hizo sin ánimo de lucro, por lo que su coste es 0. El coste del equipo de desarrollo es fijo y en caso de que se quiera contratar un nuevo servidor con una nueva licencia, se consideran los siguientes gastos.

Recurso	Coste
Ordenador	950,00 €
Licencia Windows Server	450, 00 €
Contratación de servidor	10 €/mes

Tabla 7.1: Costes

Se ha creado un diagrama de Gantt para representar gráficamente la duración estimada del proyecto.En las figuras 7.1 y 7.2 se aprecian los detalles de cada tarea, las dependencias entre tareas y la duración de cada una.

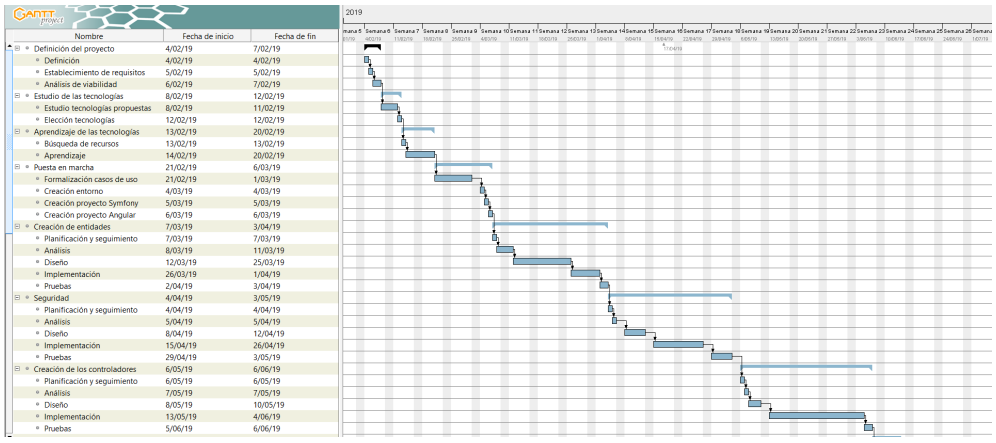

Figura 7.1: Diagrama de Gantt. Detalle 1

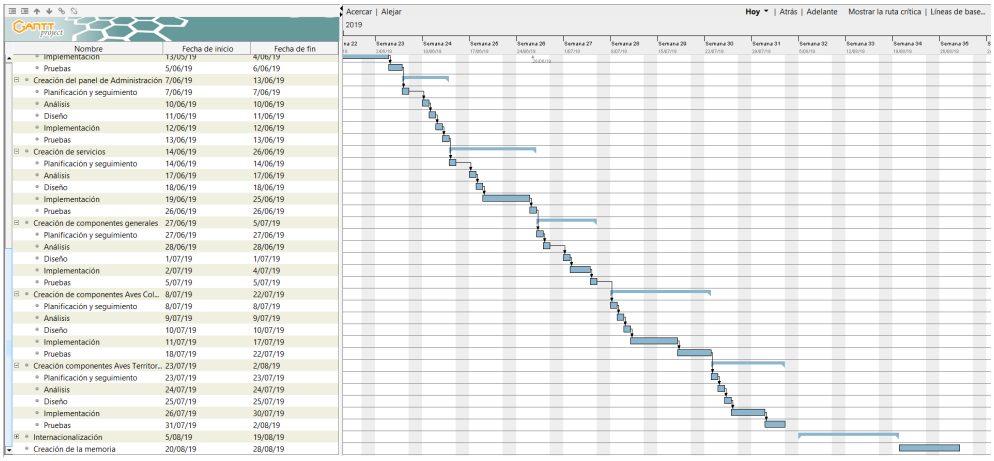

Figura 7.2: Diagrama de Gantt. Detalle 2

Teniendo en cuenta que el gasto de *ordenador* y la licencia *Windows Server* son costes fijos, se estima el coste total de la contratación del servidor. Siendo el proyecto realizado en un total de 7 meses, se estiman 70 €.

A continuación se muestra en la tabla 7.2 los costes totales.

Recursos humanos	**Coste**
Gestor proyecto	0 €
Analista	0 €
Programador	0 €
Recursos técnicos	**Coste**
Ordenador	950, 00 €
Licencia Windows Server	450,00 €
Contratación servidor	70 €
Total	***1470 €***

Tabla 7.2: Costes totales

Capítulo 8

Diseño de la aplicación

La aplicación sigue un modelo de Cliente-Servidor, por lo que tendremos dos partes bien diferenciadas: El ***front-end*** y el ***backt-end*** .
En el denominado ***front-end*** estará toda la parte que interactúa con el usuario, tanto obteniendo como devolviendo datos. Es la parte encargada de recoger los datos introducidos, procesarlos, y enviárselos al modelo de la aplicación. De la misma manera recibirá datos enviados desde el modelo, los procesará y los visualizará de cara al usuario.

El ***back-end*** o **modelo** será el encargado de recibir datos, aplicar la lógica de negocio y de ser necesario, devolver al ***front-end*** los datos procesados.

8.1 Modelos de diseño utilizados

8.1.1 Patrón MVC

El patrón **MVC** (Model-View-Controller) es un patrón en el diseño de software comúnmente utilizado para implementar interfaces de usuario, datos y lógica de control. Algunos otros patrones de diseño se basan en MVC, como **MVVM** (Model-View-Viewmodel), **MVP** (Model-View-Presenter) y **MVW** (Model-View-Whatever).

Como se mencionó en capítulos anteriores, el ***front-end*** será implementado con Angular, que no sigue un patrón MVC clásico, sino que el modelo tiene mucha relación con la vista. Esto es así ya que la forma de sincronizar los datos entre la vista y el modelo-vista es totalmente dependiente, es decir, en la vista se puede modificar el modelo y en el modelo se puede modificar la vista.

Esto hace que la independencia que se produce en un modelo-vista-controlador clásico aquí no se produzca, y por lo tanto tiende a llamarse modelo-vista vista-modelo (MVVM) o bien modelo-vista-whatever (MVW), porque no se sabe muy bien dónde identificarlo. Tam-

bién existe el modelo como lógica de negocio, como pueden ser los servicios o todo aquello que se inyecte que se puede considerar modelo, y que está totalmente independizado de la vista. Este último caso es el que aplica el sistema.

Por lo tanto, se tienen la vista y el controlador, y es en el modelo donde se aplica el concepto de Cliente-Servidor. Es el ***back-end***, o mejor denominado ***modelo***, el que interactúa con la base de datos y donde se produce la lógica de negocio.

8.1.2 Modelo Cliente-Servidor

Una forma común de organizar el software para que se ejecute en sistemas distribuidos es separar las funciones en dos partes: clientes y servidores [16].

Un ***cliente*** es un programa que utiliza servicios que otros programas brindan. Los programas que proporcionan los servicios se denominan servidores. El cliente solicita un servicio y un servidor realiza ese servicio. Las funciones del servidor a menudo requieren cierta administración de recursos, en la cual un servidor sincroniza y administra el acceso al recurso, y responde a las solicitudes del cliente con datos o información de estado. Los programas de cliente generalmente manejan las interacciones de los usuarios y a menudo solicitan datos o inician alguna modificación de datos en nombre de un usuario.

8.2 Diseño del Modelo

8.2.1 Diseño de entidades

En esta sección de detallarán las entidades persistentes que se han diseñado. En la figura 8.1 se muestra un UML autogenerado con la herramienta *yuml*, que nos representa gráficamente todas las entidades con sus atributos y sus relaciones.

Principales entidades

- **Colonia** Representa los datos que queremos guardar de una colonia.
- **Territorio** Representa los datos que queremos guardar de un territorio.
- **LocNidosCol** Representa las diferentes localizaciones que nos podemos encontrar de varios nidos en una colonia.
- **LocNidosNoCol** Representa la localización o localizaciones de un nido en un territorio.

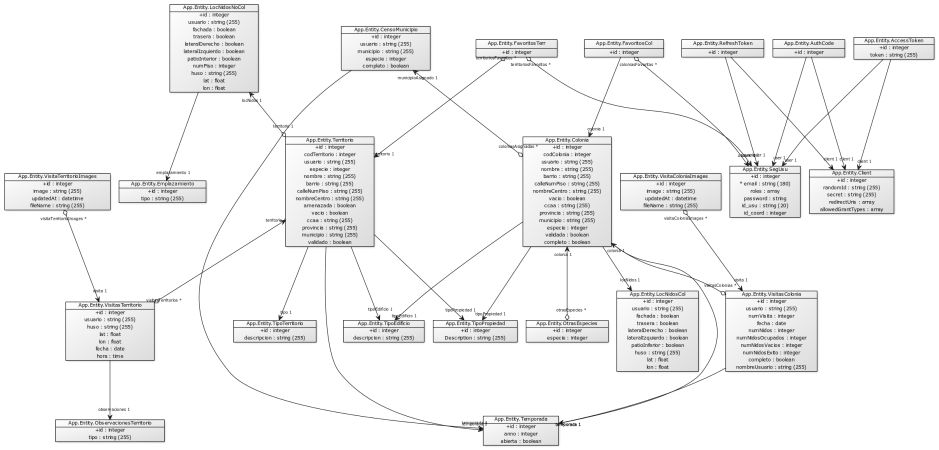

Figura 8.1: Diagrama de entidades

- **SegUsu** Representa información almacenada de un usuario. Se guardará un id único que identifica cada tupla inequívocamente, y también guardaremos un id de usuario correspondiente al identificador de la entidad de usuario almacenada en la BBDD de SEO. De los datos personales del usuario sólo se guardarán su email y su contraseña encriptada, necesarios para obtener un código de autorización. También se guardará un array con los diferentes roles que posea el usuario, y se dejará un campo llamado id-coord, que lo utilizará SEO para conocer el nivel de coordinación del usuario.

- **VisitasColonia** Representa una visita registrada por un usuario en una colonia ya creada.

- **VisitaColoniaImages** Representa la imagen subida por un usuario asociada a una visita en una colonia. Está diseñado para que en un futuro puedan subirse más fotos por visita.

- **VisitasTerritorio** Representa una visita registrada por un usuario en un territorio ya creado.

- **VisitaTerritorioImages** Representa la imagen subida por un usuario asociada a una visita en un territorio. Está diseñado para que en un futuro puedan subirse más fotos por visita.

- **CensoMunicipio** Clase que representa el censo asignado a un municipio. Para indicar qué colonias lo conforman y si entre todas ellas forman un censo completo o no.

Entidades con datos predefinidos

- **ObservacionesTerritorio** Información para complementar la visita a un territorio.

- **Temporada** Listado de temporadas disponibles, pudiendo indicar si están abiertas o cerradas.
- **TipoEdificio**

  Listado con tipos de edificios.
- **TipoPropiedad**

  Listado con tipos de propiedades.
- **TipoTerritorio**

  Información importante para complementar la creación de un territorio.
- **Emplazamiento** Información para complementar la localización del nido de un territorio.

Entidades de autenticación y autorización

- **RefreshToken** Clase necesaria para el framework de FOSOauth. Necesario en caso de que se utilice la caducidad en los códigos de autorización de los usuarios.
- **AccessToken**

  Clase necesaria para el framework de FOSOauth, encargada de guardar los códigos de acceso de los usuarios.
- **AuthCode**

  Clase necesaria para el framework de FOSOauth en caso de utilizar códigos de autorización.
- **Client**

  Clase necesaria para el framework de FOSOauth. Aquí guarda las credenciales de identifican la aplicación para poder obtener códigos de autorización.

8.2.2 Acceso a datos

Acceso a diferentes bases de datos

Se hace uso de los EntityManagers, el punto de acceso central a la funcionalidad ORM. Utiliza el patrón fachada para acceder a todos los diferentes subsistemas ORM. En este caso es necesario utilizar diferentes EntityManagers porque se accede a dos bases de datos completamente diferentes, por lo que en una se podría manejar un set completo de entidades totalmente diferente de la otra.

Por lo tanto, se configura un EntityManager con conexión a la base de datos de SEO, llamado ”seo”; y otro EntityManager con conexión a la base de datos del sistema, llamado ”default”.

Aplicación del patrón DAO

Se utiliza el patrón DAO (Data Access Object) para abstraer y encapsular todo el acceso a la base de datos. Este patrón gestiona la conexión con la base de datos para obtener, modificar y almacenar información. Para aplicar este patrón se crea una clase para cada entidad que hereda de ServiceEntityRepositorio, gracias a la cual se tienen los siguientes métodos disponibles para todas las entidades:

```
-find($id, $lockMode = null, $lockVersion = null)
-findOneBy(array $criteria, array $orderBy = null)
-findAll()
-findBy(array $criteria, array $orderBy = null, $limit = null,
$offset = null)
```

Para buscar por id, encontrar una entidad por diferentes criterios, encontrarlas todas o encontrar varias por diferentes criterios respectivamente.

Aún así, hay ciertas clases que implementan otros métodos de acceso más específicos. Son las siguientes:

- **Colonia**

```
    -findByRadius($radius, $lat, $long, $especie);
    -statAnno($especie, $anno);
    -countAnno($anno);
    -statCcaa($especie, $anno, $ccaa);
    -statProvincia($especie, $anno, $ccaa, $prov);
    -stats($temporada, $ccaam $provincia, $especie);

```

- **Territorio**

```
    -findByRadius($radius, $lat, $long, $especie);
    -statAnno($especie, $anno);
    -countAnno($anno);
    -statCcaa($especie, $anno, $ccaa);
    -statProvincia($especie, $anno, $ccaa, $prov);
    -stats($temporada, $ccaam $provincia, $especie);

```

- **VisitasColonia**

```
    -statAnno($especie, $anno);
    -statCcaa($especie, $anno, $ccaa);
    -statProvincia($especie, $anno, $ccaa, $prov);
    -statMunicipio($especie, $temporada, $ccaa, $prov, $mun);
    -statTipoEdificio($especie, $temporada, $ccaa, $prov, $mun,
    $tipo);
    -statTipoPropiedad($especie, $temporada, $ccaa, $prov, $mun,
    $tipo);

```

- **VisitasTerritorio**

```
    -stats($especie, $temporada, $ccaa, $prov, $mun, $tipo);

```

8.2.3 Servicios

Como se mencionó en capítulos anteriores, los servicios contienen los métodos necesarios para ponerse en contacto con el servidor y gestionar los datos. Se tienen 5 servicios con operaciones bien diferenciadas según con qué estructura estamos tratando. A continuación se muestra una lista de los servicios y una breve descripción sobre las tareas que realizan:

- **AuthService**: Operaciones sobre usuarios, por ejemplo conocer si está registrado, identificación...
- **ColoniasService**: Operaciones relacionadas con la estructura de las colonias. Recuperar colonias, obtener estadísticas, registrar visitas...
- **TerritoriosService**: Operaciones relacionadas con la estructura de los territorios. Recuperar colonias, obtener estadísticas, registrar visitas...
- **SharedServicesService**: Servicio creado para descargar información en formato excel. Este servicio no toma contacto con el servidor.
- **SeoApisService**: Operaciones relacionadas con la base de datos de SEO. Obtener listados de aves, listados de provincias...

8.2.4 Permisos de usuario

Con respecto a los permisos de usuario, se utilizará un sistema basado en roles. Un rol es una cadena de texto asignada a los usuarios y servirá para saber a qué tiene acceso el usuario y a qué no. Se diferencian dos roles:

- **ROL-USER**

  Usuario común. Puede hacer consultas sobre todas las tablas disponibles, pero sólo hacer cambios en algunas de ellas. Por ejemplo, hay algunas tablas para cubrir campos predefenidos, en este caso el usuario podrá consultarlas pero no modificarlas. En otros casos como por ejemplo, un usuario que crea una colonia, éste podrá consultar las colonias que registraron los demás, pero sólo podrá modificar datos de la que creó él. No tiene permisos para descargar información en formato excel de la web, ni puede entrar en la página de Administración.

- **ROL-ADMIN**

  Usuario administrador. Puede consultar, crear y modificar cualquier dato de la base de datos. Posee todos los privilegios de un usuario normal, y a mayores, provilegios de administración. Esto significa que podrá hacer todas las acciones que puede hacer un usuario normal. Es el único que puede acceder al panel de administración, y también será el único que pueda descargar información en formato excel. Podrá registrarse en la aplicación como un usuario normal, pero para ser administrador tendrá que ponerse en contacto con SEO, y la asociación decidirá si le concede el privilegio o no.

8.2.5 Controladores

Un controlador [17] es una función PHP que lee información de una petición y crea y devuelve un objeto Respuesta. La respuesta podría ser una página HTML, JSON, XML, una descarga de archivo, una redirección, un error 404 o cualquier otra cosa. El controlador ejecuta la lógica que la aplicación necesita para representar el contenido de una página.

Se han creado cuatro controladores con funciones diferenciadas, son los siguientes:

- **ColoniaController**

  Contiene la lógica de cualquier petición relacionada de alguna manera con una Colonia. Ya sea esta misma entidad, visitas, nidos, estadísticas...

- **TerritorioController** Contiene la lógica de cualquier petición relacionada de alguna manera con un territorio. Ya sea esta misma entidad, visitas, nidos, estadísticas...

- **SecurityController**

  Contiene funciones relacionadas con la gestión de usuarios, como puede ser identifiación, registro en la aplicación...

- **SeoApisController**

  Contiene la lógica necesaria para interactuar con la base de datos de SEO.

8.2.6 Rutas

Para la creación de las APIs que ofrecerá el servidor, se ha utilizado ApiPlatform [18], un framework que permite crear CRUD APIs en apenas unos minutos. Se explicará en más profundidad en el capítulo de Implementación, pero en el material adicional A se puede ver un listado de las apis disponibles para la aplicación.

8.2.7 Seguridad

En esta aplicación se utiliza el SecurityBundle de Symfony. Este componente proporciona un sistema de seguridad completo para la aplicación web. Incluye facilidades para la autenticación mediante autenticación básica HTTP, inicio de sesión de formulario interactivo o inicio de sesión de certificado X.509, pero también permite implementar estrategias propias de autenticación. Además, el componente proporciona formas de autorizar usuarios autenticados en función de sus roles.

Se pueden utilizar diferentes proveedores o "*providers*", es decir, diferentes formas de autenticación como las mencionadas anteriormente. El "*provider*" que se decidió utilizar en esta aplicación por su simplicidad es el FormLogin Authentication Provider.

Gracias a este método se dispone de un formulario de identificación y su petición HTTP de tipo POST, todo gestionado automáticamente sin que requiera apenas hacer cambios.

Se necesitará una clase que añada un poco de la lógica necesaria para este proyecto, como indicar la plantilla de Twig que se va a utilizar, o indicar cuál es la clase Usuario, necesaria para el sistema de seguridad. Esta clase se encuentra en la carpeta Security, llamada LoginFormAuthenticator. Su funcionamiento se explicará en profundidad más adelante.

8.2.8 Administración

Para la sección de administración se utilizará EasyAdminBundle [19], un framework que crea automáticamente una página visual para gestionar la base de datos. Es fácilmente gestionable y personalizable, y se utilizará para ofrecer a los usuarios administradores una manera sencilla de administrar la base de datos.

8.3 Diseño de la Vista y el Controlador

8.3.1 Plantilla

Para facilitar el diseño del "*Dashboard*" se utilizó la plantilla de uso gratuito SB-Admin-BS4-Angular6, disponible actualmente para Angular 8. Es una plantilla gratuita y de código abierto, cuya descarga se puede realizar desde la plataforma GitHub.

8.3.2 Componentes

Cada componente representa una página de la aplicación web. No son páginas independientes, en realidad todos los componentes forman parte de una única página. Lo que hace Angular es cargar en la página principal diferentes ”subpáginas” o componentes, evitando así cargas innecesarias.

A continuación se especifica el listado de los componentes que conforman la aplicación, y una breve descripción de ellos:

- **Login**: Página inicial donde el usuario introduce sus datos para identificarse
- **Selector**: Página donde el usuario selecciona Aves coloniales o no coloniales
- **SignUp**: Página de registro
- **NotFound**: Página que se muestra cuando no se encuentra una ruta.
- **CensoMunicipio**: Página para completar datos de un censo asignado a un municipio.
- **Components**: Carpeta donde se guardan los componentes Header (barra superior de navegación) y Sidebar(menú lateral)
- **DashboardTerr**: Menú principal de Aves no coloniales.
- **Dashboard**: Menú principal de Aves coloniales.
- **Docs**: Página donde el usuario encontrará archivos que puede descargar.
- **GeneralCol**: Página con estadísticas generales de Aves coloniales
- **GeneralTerr**: Página con estadísticas generales de Aves no coloniales.
- **RegisterCol**: Página con el formulario de registro de una colonia.
- **RegisterTerr**: Página con el formulario de registro de un territorio.
- **RegisterVisit**: Menú donde aparecerán las colonias y territorios marcados como favoritos, además de un buscador por código para hacer más rápido el proceso de enontrar la colonia/territorio deseado para registrar una visita.
- **StatisticsNests**: Estadísticas de nidos de colonias registradas.
- **StatisticsTerr**: Estaísticas de territorios registrados.
- **Statistics**: Estadísticas de colonias registradas.

- **ViewCloseCol**: Visualización de las colonias cercanas al usuario.
- **ViewCloseTerr**: Visualización del número de territorios cercanos al usuario.
- **ViewCol**: Listado de las colonias registradas en la aplicación.
- **ViewTerr**: Listado de los territorios registrados en la aplicación.
- **ViewVisitProfile**: Página donde se muestran las visitas registradas en una colonia.
- **ViewVisitProfileTerr**: Página donde se muestran las visitas que ha realizado el usuario en un territorio.
- **ViewVisits**: Página donde se visualizan las visitas registradas por el usuario, tanto en colonias como en territorios.

8.3.3 Internacionalización

Internacionalización es el proceso de diseño y preparación de la aplicación para poder ser usada en diferentes idiomas.

Para este proceso se utiliza la librería ngx-translate, que permite la carga dinámica de archivos de traducciones en la aplicación Angular. Esta capacidad de carga dinámica de archivos permite cambiar el idioma de la aplicación sin necesidad de recargar la pantalla.

Para ello se utiliza un archivo con extensión *.JSON* que contiene las cadenas de texto estructuradas para cada idioma. Son archivos que se guardan en la carpeta llamada *i18n*. Los idiomas disponibles son: Castellano, Gallego, Catalán, Vasco e Inglés.

Capítulo 9

Implementación de la aplicación

9.1 Implementación del Modelo

9.1.1 Estructura

Para estructurar este proyecto se utilizó un esqueleto autogenerado de Symfony. En los directorios principales se utilizan: "*config*", para guardar archivos de configuración con extensión "*.yaml*"; "*public*", para almacenar archivos, imágenes y cualquier otro que necesite acceso público; "*src*", que contiene el código principal del sistema; "*templates*", que almacena plantillas de Twig; y "*tests*", donde se encuentran todas las clases que se han creado para realizar tests unitarios con PHPUnit.

Esta estructura general se aprecia en la figura 9.6, y en las figuras 9.1, 9.2, 9.3, 9.4 y 9.5 se muestran ejemplos más detallados.

9.1.2 Mapeo de entidades con Doctrine

Se crean las clases PHP que se persistirán en la base de datos de MySQL a través del uso de anotaciones de Doctrine. Las anotaciones que se han utilizado son las siguientes:

- **@Entity** para definir la clase como una entidad y **@Table** para especificar el nombre de la tabla.
- **@OneToMany**,**@OneToOne** y **@ManyToOne** para especificar relaciones tipo 1:N, 1:1 y N:1 entre la propia clase y otra diferente.
- **@Column** sobre cada atributo que se quiere persistir en forma de columna, indicando el tipo de ésta.
- **@Id** y **@GeneratedValue** para autogenerar un id que normalmente sirve de clave primaria.

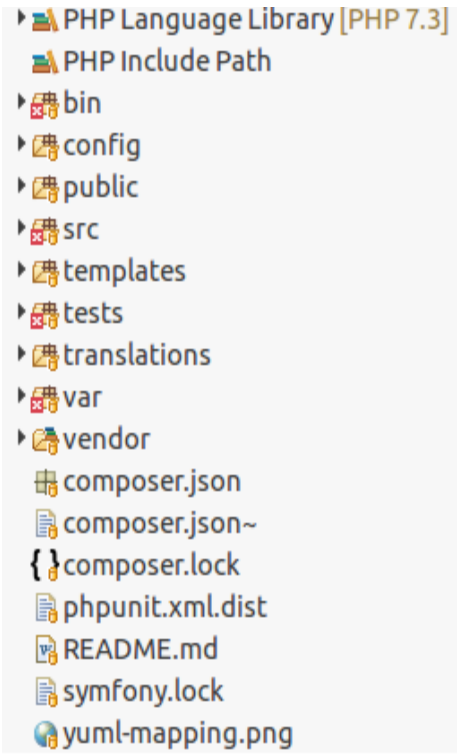

Figura 9.1: Estructura general del proyecto

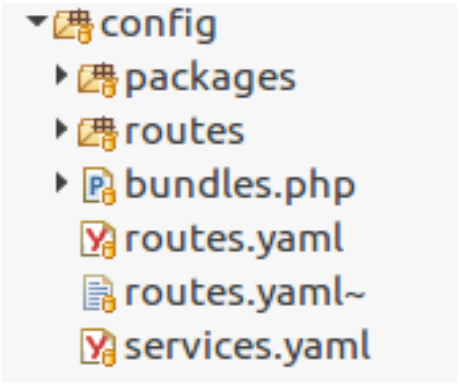

Figura 9.2: Estructura del directorio ”config”

En cada clase PHP también se aplican las anotaciones del framework de ApiPlatform, que son las siguientes:

- **@ApiResource()**. Indica que la entidad dispondrá de un CRUD Api completo y autogenerado.
- **@ApiFilter()**. Mediante esta anotación se espcifican diferentes filtros que pueden aplicarse mediante la inclusión de parámetros en la petición. Pueden ser de diferentes tipos

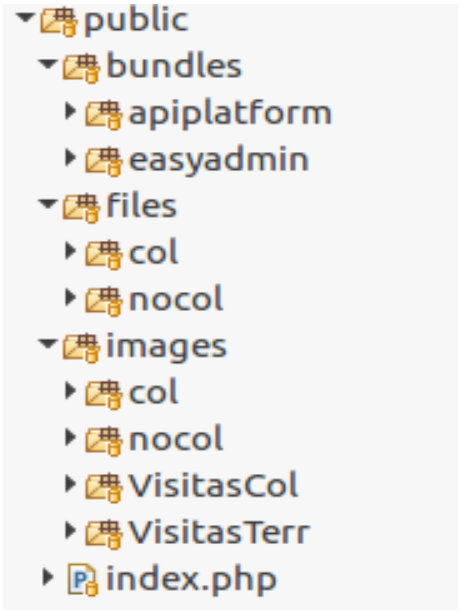

Figura 9.3: Estructura del directorio ”public”

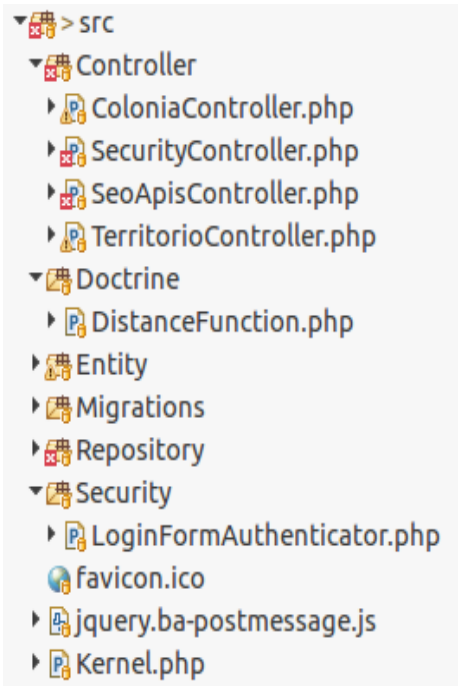

Figura 9.4: Estructura del directorio ”src”

según el tipo del atributo sobre el que aplican.

Las anotaciones del tipo ”@Groups” forman parte del serializador de Symfony, y se utilizan para indicar qué campos de la entidad se quieren devolver bajo ciertas circuns-

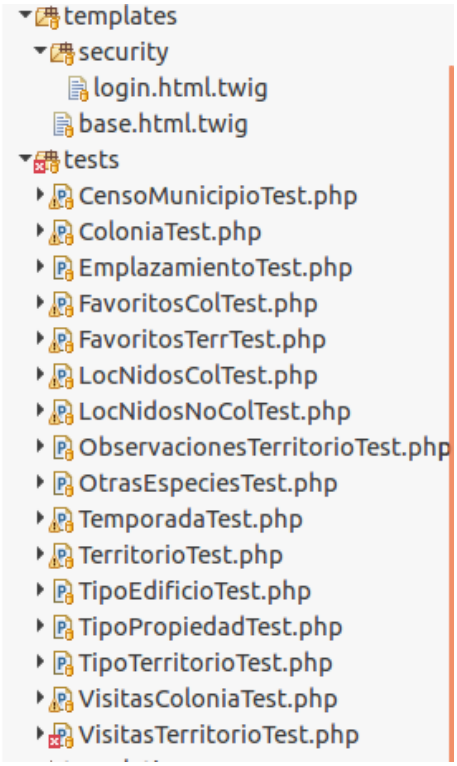

Figura 9.5: Estructura de los directorios ”templates” y ”tests”

tancias.

Función Distancia en doctrine

Es necesario tener una función que calcula la distancia entre dos puntos de la Tierra, para así poder conocer la posición del usuario y definir qué colonias o territorios están cerca de él. Se definen diferentes distancias en kilómetros para que el usuario pueda decidir si quiere ampliar o reducir el rango de búsqueda. Una vez decidida la distancia se pasarán como parámetros a esta función las coordenadas del usuario, junto con las coordenadas de la colonia o territorio y el radio de búsqueda.

Una vez definida la función debe especificarse en la configuración de Doctrine para poder usarse en un EntityManager específico, tal y como se aprecia en la figura 9.6.

Está función se aplicará para filtrar resultados como cualquier otra función SQL en el repositorio de la entidad deseada.

```yaml
doctrine:
    dbal:
        default_connection: default
        connections:
            default:
                # configure these for your database server
                url: '%env(DATABASE_URL)%'
                driver: 'pdo_mysql'
                server_version: '5.7'
                charset: utf8mb4
            seo:
                # configure these for your database server
                url: '%env(DATABASE_SEO_URL)%'
                driver: 'pdo_mysql'
                server_version: '5.7'
                charset: UTF8

    orm:
        auto_generate_proxy_classes: '%kernel.debug%'
        default_entity_manager: default

        entity_managers:
            default:
                connection: default
                dql:
                    numeric_functions:
                        mydistance: App\Doctrine\DistanceFunction
                mappings:
                    App:
                        is_bundle: false
                        type: annotation
                        dir: '%kernel.project_dir%/src/Entity'
                        prefix: 'App\Entity'
                        alias: App
            seo:
                connection: seo
                mappings:
                    Seo:
                        is_bundle: false
                        type: annotation
                        dir: '%kernel.project_dir%/src/Entity'
                        prefix: 'App\Entity'
                        alias: Seo
```

Figura 9.6: Extracto de doctrine.yaml

9.1.3 Seguridad

Para todo lo relacionado con la seguridad se utiliza el componente de seguridad de Symfony. A partir de este componente se puede configurar un firewall con numerosas funcionalidades, entre ellas desteaca la denegación de acceso, ya sea a usuarios registrados o sin registrar.

Para el panel de administración implementado con EasyAdmin es necesario disponer de un sistema de autenticación de usuarios que permita proteger esta sección tan delicada.

Hay numerosas formas de realizar esta autenticación, pero se ha decidido por crear un formulario de *login* con el componente de Guard, un sistema de autenticación ideal para trabajar con formularios de este tipo.

Se crea una clase personalizada que extiende de *AbstractFormLoginAuthenticator*, y se realizan pequeños cambios para adecuar la clase al sistema. También se crea la plantilla de Twig que se utilizará para el formulario de *login*. En la figura 9.7 se aprecia un esquema con el flujo que sigue este sistema.

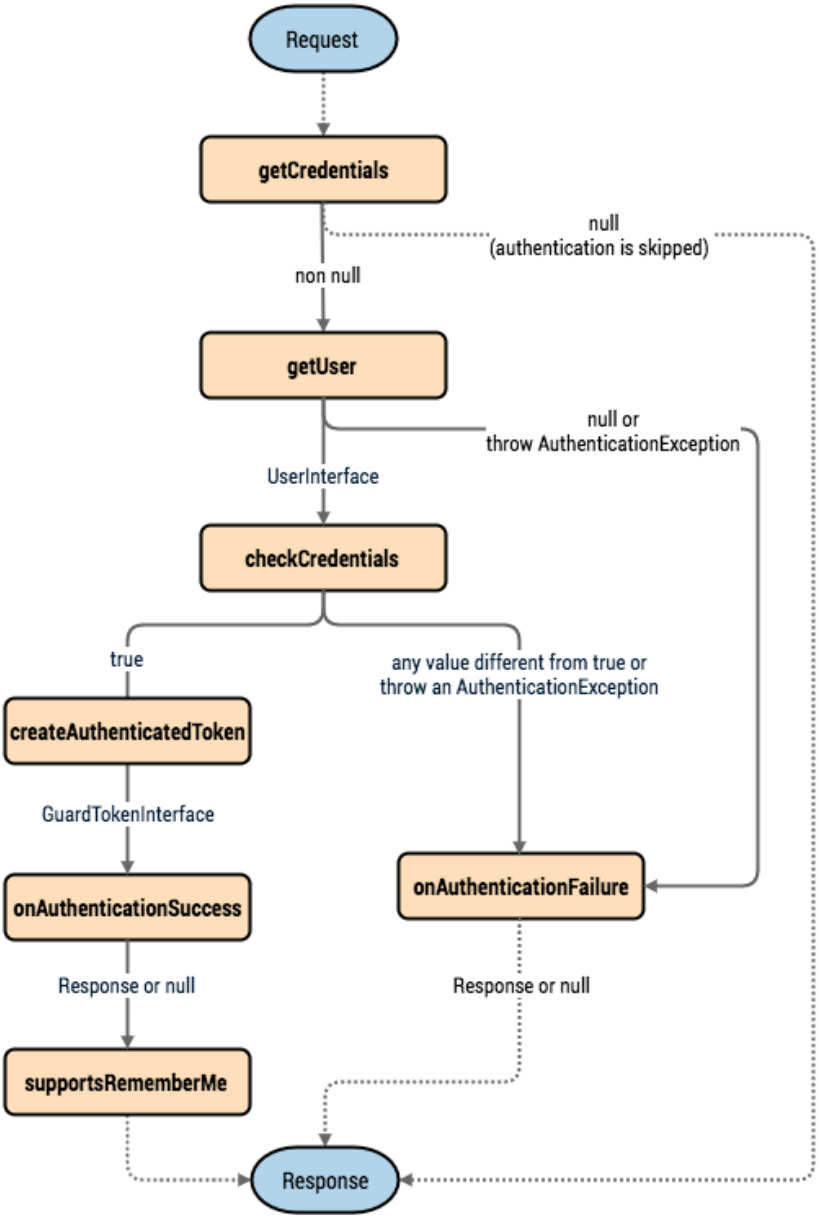

Figura 9.7: Flujo de la autenticación

Por otra parte, también se ha implementado el protocolo OAuthv2 con el framework FOSOauth. Este componente gestiona toda la generación, obtención y eliminación de API Tokens,

necesarios para la autorización de cada usuario.

Para el correcto funcionamiento de este framework es necesario crear y persistir las siguientes clases: *Client* ,*AccessToken*,*RefreshToken* y *AuthCode*.

En la figura 9.8 se observa el flujo de autorización del protocolo OAuthv2.

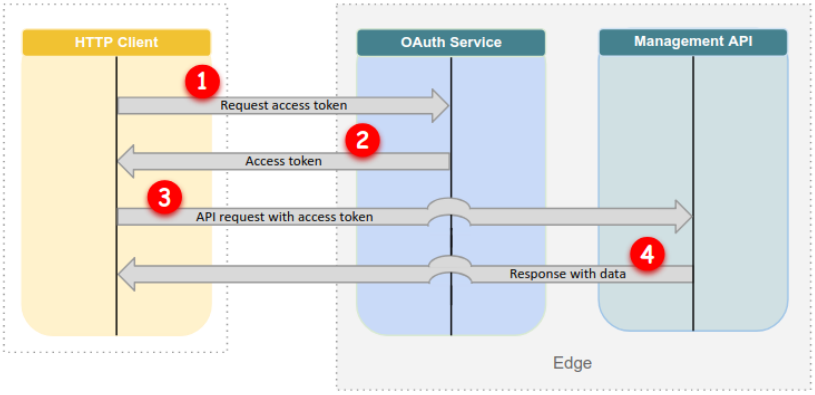

Figura 9.8: Flujo de la autorización

9.1.4 Controladores

Los controladores son las clases que aplican la lógica de negocio. Si se quiere utilizar la función de un controlador específico para una ruta específica, se configurará en el fichero *routes.yaml*.

```
api_provincias:
    path: '/api/provincias/{id}'
    methods: ['GET']
    defaults:
        _controller: App\Controller\SeoApisController::provincias
```

Figura 9.9: Ejemplo de ruta configurada para trabajar con un controlador

La clase controlador siempre extenderá de *Controller*, y a partir de aquí se podrán generar todas las funciones que se consideren necesarias. Cada función puede recibir el elemento *Petición*, así como todos los parámetros especificados en la ruta solicitada.

Una vez aplicada la lógica de negocio las funciones pueden devolver respuestas con los datos necesarios, o incluso lanzar excepciones. Las más utilizadas en este sistema son ***NotFoundException*** e ***InvalidArgumentException***, ambas disponibles en el componente principal de Symfony.

En la figura 9.10 se observa un ejemplo de función que recibe una *Request*, es decir, el objeto petición. Obtiene los parámetros de la ruta, aplica lógica de negocio y finalmente devuelve

una respuesta.

```
public function getColoniasCercanas(Request $request){
    $lat = $request->query->get("lat");
    $lon = $request->query->get("lon");
    $rad = $request->query->get("rad");
    $especie = $request->query->get("especie");
    $colonias = new ArrayCollection();
    if($rad != NULL)
    {

        $colonias= $this->getDoctrine()->getRepository(Colonia::class)->findByRadius($rad, $lat, $lon, $especie);
    }

    return new JsonResponse(
        $this->normalizer->normalize(
            $colonias, 'json', ['groups' => ['colonia']]
    ));
}
```

Figura 9.10: Ejemplo de función de un controlador

9.1.5 Administración con EasyAdmin

El panel de Administración se ha implementado con el framework de EasyAdmin, un conjunto de herramientos que facilitan la creación de este tipo de paneles.

Este framework se personaliza mediante su hoja de configuración, en la se deberán indicar las entidades que se quieren administrar. También se pueden personalizar pequeños detalles en este mismo fichero de configuración, como por ejemplo los colores. EasyAdmin permite modificar cada una de las plantillas Twig que utiliza para renderizar la vista, pero para este proyecto sólo ha sido necesario modificar algunos colores por lo que no ha sido necesario modificar ninguna plantilla.

9.2 Implementación de la Vista y el Controlador

9.2.1 Estructura

Para estructurar esta parte se utilizó un esqueleto autogenerado de Angular. En los directorios principales se utilizan: "*app*", para los componentes de la aplicación; "*assets*", para guardar script personalizados y los ficheros de idiomas;y "*services*", que contienen los servicios.

Esta estructura general se aprecia en la figura 9.11, y en las figuras se muestran ejemplos más detallados.

9.2.2 Componentes

Los componentes forman las diferentes pantallas que el usuario visualiza. Un componente por lo general estará formado por un fichero de tipo HTML, fichero de estilos, normalmente con extensión "*.scss*" y el fichero typescript que se corresponde con el controlador.

El principio de cada controlador se anotará con "*@Component*", para indicarle cuál es la vista, qué fichero de estilos va a aplicar, y qué nombre se le da al componente para poder

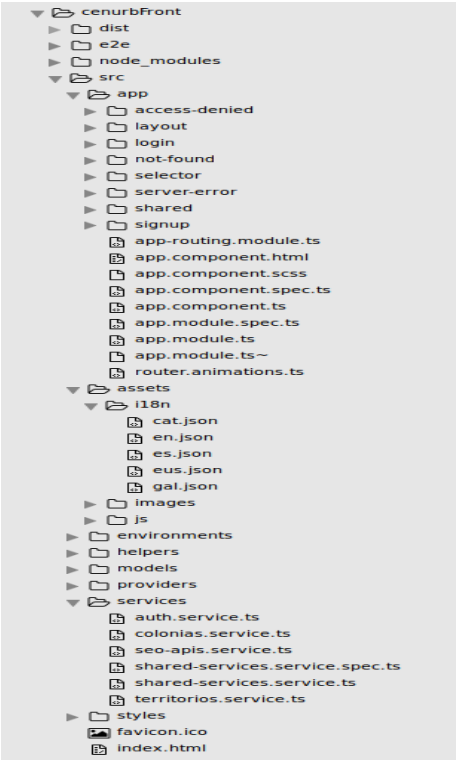

Figura 9.11: Estructura general del proyecto Angular

cargarlo correctamente. En la figura 9.12 se puede ver un ejemplo de esta anotación.

```
@Component({
  selector: 'app-statistics',
  templateUrl: './statistics.component.html',
  styleUrls: ['./statistics.component.scss']
})
```

Figura 9.12: Anotación en el componente

Las rutas de navegación estre los diferentes componentes se especifican en los archivos de "*routing*", pueden estar dentro de cada componente o utilizar un archivo general para todas las rutas. Dentro de estos ficheros se especificará el nombre de la ruta que se tiene que seguir y el componente que ésta carga.

En ciertos componentes se realizó una integración con Google Maps para poder ofrecer ciertas funciones. Para el sistema de Angular se instaló la librería *agm-core*, que permite integrar la API de Google Maps con anotaciones de Angular. Se puede ver un ejemplo en la figura

9.13, donde se crea un mapa centrado en un punto específico, además de pintar un círculo alrededor de unas coordenadas ofrecidas por el controlador.

```
<agm-map [zoom]="12" [latitude]="latitude" [longitude]="longitude">
    <agm-circle [latitude]="latitude" [longitude]="longitude" [radius]="radio" [fillColor]="'red'"
      [circleDraggable]="false"
      [editable]="false">
    </agm-circle>
</agm-map>
```

Figura 9.13: Creación de un mapa con anotaciones de Angular

9.2.3 Servicios

Para poder utilizar los servicios estos se deben inyectar en el constructor del componente que desea utilizarlos. Una vez inyectados se podrá acceder a todas las funciones que ofrecen.

Normalmente ofrecerán un método por cada petición HTTP que se desee hacer al servidor. En la figura 9.14 se observa el ejemplo de dos funciones de un servicio. Realizan peticiones diferentes a diferentes apis.

```
//Registra una nueva colonia
nuevaColonia(colonia: Colonia) {
  let config = {headers: new HttpHeaders().set("Content-Type", 'application/json')};

  let response=this.http.post(this.url + '/api/colonias', JSON.stringify(colonia), config);
  return response;
}

//Recupera las colonias cercanas a la posicion del usuario, con un radio de distancia
recuperaColoniasCercanas( radio, lat, lon, especie) {
  return this.http.get<any>(this.url + '/api/closeCol?rad=' + radio + '&lat=' + lat + '&lon=' + lon + '&especie=' + especie);
}
```

Figura 9.14: Ejemplo de servicios

9.2.4 Internacionalización

La internacionalización se implementó mediante la librería ngx-translate. Permite mediante anotaciones de Angular obtener un texto de un archivo estructurado, típicamente en formato JSON. También permite la carga dinámica de estos archivos, por lo que se puede cambiar de idioma sin necesitad de recargar la vista de cara al usuario. Es escalable, por lo que se podrán añadir todos los idiomas que se deseen en el futuro.

Capítulo 10

Pruebas

En este capítulo se explicarán las pruebas que se realizaron sobre el sistema .
Las pruebas son básicamente un conjunto de actividades dentro del desarrollo de software. Dependiendo del tipo de pruebas, estas actividades podrán ser implementadas en cualquier momento de dicho proceso de desarrollo.

Los objetivos de las pruebas son los siguientes:

- Probar si el software no hace lo que debe.
- Probar si el software hace lo que no debe, es decir, si provoca efectos secundarios adversos.
- Descubrir un error que aún no ha sido descubierto.
- Encontrar el mayor número de errores con la menor cantidad de tiempo y esfuerzo posibles.
- Mostrar hasta qué punto las funciones del software operan de acuerdo con las especificaciones y requisitos del cliente.

10.1 Pruebas de unidad

Una prueba unitaria es una forma de comprobar el correcto funcionamiento de una unidad de código. Es importante que estas pruebas sean automáticas e indepiendientes entre sí, es decir, cada prueba debe crear, modificar o destruir sus propios datos sin afectar a las demás.

En esta aplicación se ha creado una clase de pruebas para cada entidad y se han guardado en la carpeta de *tests*. Se ejecutarán mediante PHPUnit, un framework orientado a pruebas para cualquier código PHP.

Se ejecutan todos los tests mediante el comando

```
1    php /bin/phpunit tests
```

y se comprueban los resultados. En la figura 10.1 se aprecian que todos los resultados han sido correctos.

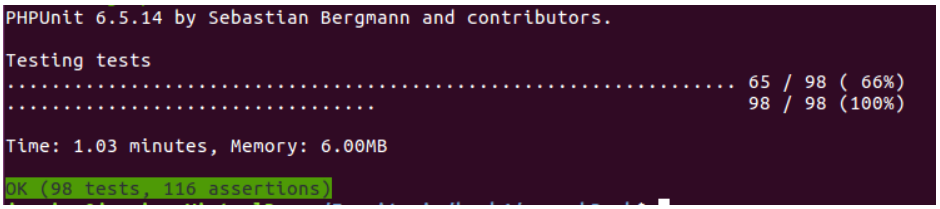

Figura 10.1: Pruebas de unidad satisfactorias

10.2 Pruebas de integración

El objetivo de las pruebas de integración es verificar el correcto ensamblaje entre los distintos componentes una vez que han sido probados unitariamente con el fin de comprobar que interactúan correctamente a través de sus interfaces .

En un primer paso se creó un paquete de peticiones HTTP para probar con cualquier cliente que pueda realizar este tipo de peticiones. Se ha utilizado Postman por su versatilidad y facilidad de uso.

Se realizaron diferentes peticiones a cada API para comprobar su correcto funcionamiento, y se llegó a la conclusión de que todas cumplían con lo esperado.

Una vez creado todo el sistema se realizaron las mismas peticiones pero desde la interfaz de usuario, para así comprobar el correcto funcionamiento de todos los componentes juntos.

10.3 Pruebas del modelo visual

Se comprobaron que todos los componentes funcionasen juntos correctamente. Como el sistema sigue un modelo *responsive* se ha probado toda la interfaz en diferentes navegadores y con diferentes vistas. Gracias al depurador de *Firefox* o *Chrome* se pudo probar la interfaz simulando diferentes dispositivos, como tablets o móviles de diferentes marcas y tamaños.

Todas las pruebas fueron correctas y satisfactorias.

Capítulo 11

Conclusiones y futuras líneas de trabajo

El sistema cumple todos los objetivos establecidos. La apariencia ha seguido la línea de otras aplicaciones de SEO, aplicando colores y módulos de estilo similar.

La gran ventaja de utilizar un diseño *responsive* es que la aplicación está orientada a utilizarse desde dispositivos móviles, por lo que la interfaz gráfica se adapta perfectamente a estas necesidades.

El componente de seguridad resultó ser el más complejo, pues su diseño implicaba autenticar usuarios que no existían en la base de datos gestionada con Symfony. La autorización también resultó compleja por este mismo motivo, pero finalmente se llevó a cabo satisfactoriamente.

Se aprendieron nuevas tecnologías muy interesantes como el protocolo OAuthv2 o el framework de Doctrine, que resultó ser sencillo pero al mismo tiempo completo.

También resultó interesante el uso de librerías de Angular para integrar la Vista con Google Maps. Resultó ser muy sencillo utilizar anotaciones de este framework para integrar esta API de Google.

El principal objetivo que se persiguió fue la creación de una aplicación con la que el cliente quedara satisfecho, ofreciendo la mayor calidad posible.

En cuanto a **líneas futuras de trabajo**, se consideron varios aspectos mejorables o con previsión de cambio en el futuro. A continuación se enumeran estos aspectos:

- Mejora de aspectos visuales. Hay ciertos elementos que pueden mejorarse para ofrecer

una mejor interfaz gráfica.

- Subida de varias fotografías por visita. Actualemente sólo se permite añadir una imagen en cada visita.
- Añadir o quitar especies. El sistema está diseñado para que no represente un problema cambiar los listados de aves, pudiendo éstos aumentar o disminuir.

Apéndices

Apéndice A

Material adicional

CensoMunicipio

POST	/api/censo-municipios
GET	/api/censo-municipios
GET	/api/censo-municipios/{id}
PUT	/api/censo-municipios/{id}
DELETE	/api/censo-municipios/{id}

Colonia

GET	/api/closeCol
POST	/api/colonias
GET	/api/colonias
GET	/api/colonias/favoritos/{id}
GET	/api/colonias/{id}
DELETE	/api/colonias/{id}
POST	/api/colonias/{id}/loc-nidos
POST	/api/colonias/{id}/otras-especies
POST	/api/colonias/{id}/visitas
GET	/api/especies/stats
GET	/api/especies/{id}/statsAnno
GET	/api/especies/{id}/statsAnnoCol
GET	/api/especies/{id}/statsCcaa
GET	/api/especies/{id}/statsCcaaCol
GET	/api/especies/{id}/statsMunicipioCol
GET	/api/especies/{id}/statsProvincia
GET	/api/especies/{id}/statsProvinciaCol
GET	/api/especies/{id}/statsTipoEdificioCol
GET	/api/especies/{id}/statsTipoPropiedadCol

Emplazamiento

GET	/api/emplazamientos
GET	/api/emplazamientos/{id}

Territorio

GET	/api/especies/statsTerr
GET	/api/especies/{id}/statsAnnoTerr
GET	/api/especies/{id}/statsCcaaTerr
GET	/api/especies/{id}/statsObservaciones
GET	/api/especies/{id}/statsProvinciaTerr
POST	/api/territorios
GET	/api/territorios
GET	/api/territorios/favoritos/{id}
GET	/api/territorios/{id}
PUT	/api/territorios/{id}
DELETE	/api/territorios/{id}
POST	/api/territorios/{id}/loc-nidos
POST	/api/territorios/{id}/visitas

FavoritosCol

GET	/api/favoritos-cols
POST	/api/favoritos-cols
GET	/api/favoritos-cols/{id}
DELETE	/api/favoritos-cols/{id}
PUT	/api/favoritos-cols/{id}

FavoritosTerr

GET	/api/favoritos-terrs
POST	/api/favoritos-terrs
GET	/api/favoritos-terrs/{id}
DELETE	/api/favoritos-terrs/{id}
PUT	/api/favoritos-terrs/{id}

LocNidosCol

GET	/api/loc-nidos-cols
GET	/api/loc-nidos-cols/{id}
PUT	/api/loc-nidos-cols/{id}
DELETE	/api/loc-nidos-cols/{id}

VisitasColonia

GET	/api/misVisitas
POST	/api/visitas-colonias
GET	/api/visitas-colonias
GET	/api/visitas-colonias/{id}
PUT	/api/visitas-colonias/{id}
DELETE	/api/visitas-colonias/{id}

ObservacionesTerritorio

GET	/api/observaciones-territorios
GET	/api/observaciones-territorios/{id}

OtrasEspecies

GET	/api/otras-especies
GET	/api/otras-especies/{id}
PUT	/api/otras-especies/{id}
DELETE	/api/otras-especies/{id}

Temporada

GET	/api/temporadas
GET	/api/temporadas/{id}

TipoEdificio

GET	/api/tipo-edificios
GET	/api/tipo-edificios/{id}

TipoPropiedad

GET	/api/tipo-propiedads
GET	/api/tipo-propiedads/{id}

TipoTerritorio

GET	/api/tipo-territorios
GET	/api/tipo-territorios/{id}

VisitasTerritorio

POST	/api/visitas-territorios
GET	/api/visitas-territorios
GET	/api/visitas-territorios/{id}
PUT	/api/visitas-territorios/{id}
DELETE	/api/visitas-territorios/{id}

Lista de acrónimos

PHP *Hypertext PreProcessor.*

HTML *HyperText Markup Language.*

CSS *Cascading Style Sheets.*

MVC *Model-View-Controller.*

ORM *Object Relational Mapping.*

REST *Representational State Transfer.*

API *Application Programming Interface.*

CRUD *Create Read Update and Delete.*

Bibliografía

[1] "Diseño web adaptable," 2018. [En línea]. Disponible en: https://developers.google.com/search/mobile-sites/mobile-seo/responsive-design?hl=es

[2] "Modelo cliente-servidor." [En línea]. Disponible en: http://www.ite.educacion.es/formacion/materiales/157/cd/m1_1_conceptos_basicos_de_internet/modelo_clienteservidor.html

[3] U. de Alicante, "Modelo vista controlador (mvc)." [En línea]. Disponible en: https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html

[4] "Mvc in an angular world," 2019. [En línea]. Disponible en: https://scotch.io/tutorials/mvc-in-an-angular-world

[5] "Lenguajes del lado del servidor." [En línea]. Disponible en: https://www.axarnet.es/blog/lenguajes-del-lado-del-servidor/

[6] N. Martín, "Oauth 2.0: equilibrio y usabilidad en la securización de apis," 2018. [En línea]. Disponible en: https://www.paradigmadigital.com/dev/oauth-2-0-equilibrio-y-usabilidad-en-la-securizacion-de-apis/

[7] S. Docs, "Security." [En línea]. Disponible en: https://symfony.com/doc/current/security.html

[8] "¿qué es javascript?" [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/Qu%C3%A9_es_JavaScript

[9] "Html5 tutorial." [En línea]. Disponible en: https://www.w3schools.com/html/

[10] "Css tutorial." [En línea]. Disponible en: https://www.w3schools.com/css/

[11] V. Robles, "¿qué es angular?" [En línea]. Disponible en: https://victorroblesweb.es/2017/08/05/que-es-angular-y-para-que-sirve/

[12] "Manual php." [En línea]. Disponible en: https://www.php.net/manual/es/intro-whatis.php

[13] "Symfony para programadores." [En línea]. Disponible en: https://symfony.es/pagina/que-es-symfony/

[14] "Symfony." [En línea]. Disponible en: https://www.ecured.cu/Symfony

[15] "Patrón de diseño modelo-vista-controlador." [En línea]. Disponible en: https://www.ibm.com/support/knowledgecenter/es/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/csdmvcdespat.htm

[16] "The client/server model." [En línea]. Disponible en: https://www.ibm.com/support/knowledgecenter/en/SSNAQ8_10.1.0/com.ibm.cics.tx.doc/concepts/c_clnt_sevr_model.html

[17] "Controller." [En línea]. Disponible en: https://symfony.com/doc/current/controller.html

[18] "Apiplatform." [En línea]. Disponible en: https://api-platform.com/docs/core/

[19] "Easyadminbundle." [En línea]. Disponible en: https://symfony.com/doc/master/bundles/EasyAdminBundle/index.html