

16 DE JUNIO DE 2025



WECOOK – RED SOCIAL DE RECETAS

DESARROLLO DE APLICACIONES WEB

SERGIO ÁLVAREZ DE RON

SERGIO MONTOIRO

JONATHAN HIDALGO

TUTOR: HUGO MARÍA VEGAS CARRASCO

IES LUIS BRAILLE

Índice de contenidos

1.	Resumen	6
2.	Abstract.....	6
3.	Introducción	7
4.	Objetivos	8
5.	Alcance del proyecto.....	9
5.1.	Requisitos funcionales y no funcionales.....	9
	Requisitos funcionales	9
	Requisitos no funcionales	10
5.2.	Prototipado	11
	Wireframing	11
	Bocetado de vistas.....	12
5.3.	Tecnologías utilizadas	20
	Laravel.....	20
	Blade	20
	JavaScript	21
	Bootstrap	21
	MySQL.....	21
	MySQL Workbench	22
	Git	22
	GitHub.....	22
	Visual Studio Code	22
	Figma	23
	Miro	23
	Canva	23
5.4.	Diagrama de Entidad-Relación	24
5.5.	Diagrama de casos de uso	26

6.	Marco práctico	27
	Estructura	27
	Landing page con inicio de sesión y registro de usuario	31
	Recordar contraseña:.....	34
	Verificación de usuario	36
	Barra de navegación	37
	Listado de recetas:.....	38
	Detalle de receta.....	40
	Crear receta	46
	Perfil de usuario.....	47
	Recetas guardadas.....	50
	Ajustes de cuenta	51
	Pie de página	53
	Administración de recetas y usuarios.....	55
7.	Evolución del proyecto y trabajo futuro	58
	Evolución del proyecto	58
	Trabajo futuro	58
8.	Conclusiones	60
9.	Agradecimientos	60
10.	Bibliografía	61
11.	Webgrafía.....	61
12.	Anexos.....	62
	Manual técnico	62

Índice de ilustraciones

Ilustración 1 Wireframing.....	11
Ilustración 2 Vista "Landing page"	12
Ilustración 3 Vista ""Listado de recetas"	13
Ilustración 4 Vista "Perfil de usuario"	13
Ilustración 5 Vista "Detalle de receta"	14
Ilustración 6 Vista "Recetas guardadas"	14
Ilustración 7 Vista "Administración"	15
Ilustración 8 Vista "Editar cuenta"	15
Ilustración 9 Vista "Contacto"	16
Ilustración 10 Vistas de información legal.....	16
Ilustración 11 Vistas móvil "Perfil de usuario" y "Publicar/editar receta"	17
Ilustración 12 Vistas móvil "Landing page" y "Listado de recetas"	18
Ilustración 13 Vista móvil "Detalle de receta"	19
Ilustración 14 Logo de Laravel	20
Ilustración 15 Logo de Blade	20
Ilustración 16 Logo de JavaScript	21
Ilustración 17 Logo de Bootstrap	21
Ilustración 18 Logo de MySQL	21
Ilustración 19 Logo MySQL Workbench.....	22
Ilustración 20 Logo de Git.....	22
Ilustración 21 Logo de GitHub	22
Ilustración 22 Logo de VS Code	22
Ilustración 23 Logo de Figma.....	23
Ilustración 24 Logo de Miro.....	23
Ilustración 25 Logo de Canva.....	23
Ilustración 26 Diagrama entidad-relación	24

Ilustración 27 Esquema de la base de datos	25
Ilustración 28 Diagrama de casos de uso	26
Ilustración 29 Estructura de carpetas de la aplicación	27
Ilustración 30 Sección principal de la landing page.....	31
Ilustración 31 Secciones secundarias de la landing page.....	31
Ilustración 32 Ruta raíz.....	32
Ilustración 33 Formulario de registro con error de validación de correo electrónico.....	32
Ilustración 34 Formulario de registro con error de validación de contraseña	33
Ilustración 35 Función que maneja redirección de usuarios registrados.....	33
Ilustración 36 Controlador de usuarios con función de registro de usuarios	33
Ilustración 37 Formulario de recordar contraseña.....	34
Ilustración 38 Correo recibido para recuperar contraseña	34
Ilustración 39 Formulario de restablecer contraseña	35
Ilustración 40 Funciones que manejan la recuperación de contraseña	35
Ilustración 41 Formulario con error de validación de correo electrónico.....	35
Ilustración 42 Correo recibido de verificación de usuario registrado	36
Ilustración 43 Función que verifica y redirige al usuario.....	36
Ilustración 44 Plantilla que maneja las vistas y secciones de la aplicación	37
Ilustración 45 Listado de recetas.....	38
Ilustración 46 Controlador de recetas con función que muestra el listado de recetas.....	39
Ilustración 47 Detalle de receta y comentarios y respuestas de usuarios	40
Ilustración 48 Funciones de editar y actualizar receta	41
Ilustración 49 Función de eliminar receta	41
Ilustración 50 Pop-up de doble confirmación para eliminar receta.....	42
Ilustración 51 Pop-up con confirmación de receta eliminada.....	42
Ilustración 52 Función de guardado de receta	43
Ilustración 53 Función de eliminar receta	43

Ilustración 54 Función para marcar recetas gustadas	44
Ilustración 55 Función para eliminar marca de recetas gustadas	44
Ilustración 56 Función para comentar receta.....	45
Ilustración 57 Función para responder a un comentario	45
Ilustración 58 Formulario de creación/edición de receta	46
Ilustración 59 Función de guardado de receta	46
Ilustración 60 Perfil de usuario.....	47
Ilustración 61 Función para ver perfil de usuario.....	48
Ilustración 62 Función para ver seguidores y seguidos de usuario	48
Ilustración 63 Seguidores/seguídos de usuario.....	48
Ilustración 64 Función de actualizar perfil de usuario	49
Ilustración 65 Formulario de edición de perfil de usuario	49
Ilustración 66 Lista de recetas guardadas	50
Ilustración 67 Función para listar recetas guardadas	50
Ilustración 68 Formulario de ajustes de usuario	51
Ilustración 69 Ruta al formulario de ajustes de usuario.....	51
Ilustración 70 Funciones para actualizar en ajustes de usuario	52
Ilustración 71 Pie de página de la web.....	53
Ilustración 72 Vistas de información legal de la plataforma	53
Ilustración 73 Formulaio de contacto con soporte técnico	54
Ilustración 74 Función que maneja envío de correo de contacto	54
Ilustración 75 Administración de recetas	55
Ilustración 76 Administración de usuarios	56
Ilustración 77 Ruta a vista de administración.....	56
Ilustración 78 Función que lista tabla de recetas para administración	56
Ilustración 79 Función que lista tabla de usuarios para administración	57

1. RESUMEN

Este proyecto consiste en el desarrollo de una aplicación web, concretamente una red social para compartir recetas de cocina. El foco principal ha sido construir una plataforma donde los usuarios puedan compartir recetas y mensajes e interactuar con otros usuarios de forma cómoda, sencilla y segura.

Para ello, hemos empleado Laravel en su versión 12, un framework de PHP moderno y robusto, que ha facilitado la implementación de funcionalidades en nuestra aplicación, como la autenticación de usuarios, manejo de rutas, controladores, relación entre modelos, validación de formulario y un largo etcétera de estándares para aplicaciones web.

A través de este proyecto hemos podido aprender la estructura de frameworks como Laravel, así como la forma de trabajar con él de forma eficiente. Además, hemos podido aplicar los conocimientos aprendidos a lo largo del grado, desde el manejo y estructuración de base de datos, hasta el diseño visual de la web, pasando por la creación de las diferentes funcionalidades a través de la programación.

2. ABSTRACT

This project involves the development of a web application, specifically a social network for sharing cooking recipes. The main focus was to build a platform where users can share recipes and messages and interact with other users in a convenient, simple, and secure way.

To achieve this, we used Laravel version 12, a modern and robust PHP framework, which facilitated the implementation of features in our application, such as user authentication, route management, controllers, model relationships, form validation, and a long list of web application standards.

Through this project, we were able to learn the structure of frameworks like Laravel, as well as how to work with it efficiently. In addition, we were able to apply the knowledge acquired throughout the degree, from database management and structuring to the visual design of the website, including the creation of different functionalities through programming.

3. INTRODUCCIÓN

En este trabajo se propone el diseño y desarrollo de una red social centrada en compartir recetas de cocina empleando tecnologías de desarrollo web como Laravel, JavaScript y MySQL. El auge y relevancia de las redes sociales en la actualidad nos ha llevado a crear una red social dedicada a la gastronomía, más concretamente sobre recetas, pues es un tema que a los integrantes del grupo de trabajo nos interesa. Esta plataforma busca ofrecer un espacio donde los usuarios puedan intercambiar sus mejores recetas, fotos y consejos, fomentando una experiencia interactiva y colaborativa.

El desarrollo de esta aplicación se sustenta en el uso de Laravel, un framework de PHP que gestiona el backend y la lógica de negocio, además de facilitar plantillas Blade para manejar el frontend de forma sencilla junto a JavaScript. Por otro lado, MySQL es la base de datos elegida para almacenar la información de los usuarios, recetas y comentarios, así como todas sus relaciones.

Todo esto se ha planteado con la finalidad de ofrecer una solución práctica, escalable y sencilla para los amantes de la gastronomía o cualquier persona que quiera aprender a cocinar o necesite de recursos culinarios para el día a día, además de incluir funciones que fomentan la interacción social.

4. OBJETIVOS

- Desarrollar una aplicación web funcional que permita a los usuarios registrarse, iniciar sesión y gestionar su cuenta de manera segura y sencilla, utilizando Laravel como framework.
- Implementar un sistema para la publicación de recetas de cocina, en el que los usuarios puedan crear, editar y eliminar sus propias recetas incluyendo título, descripción, ingredientes, pasos de preparación y una imagen ilustrativa.
- Diseñar una interfaz web intuitiva, atractiva y responsive, garantizando una experiencia de usuario óptima en dispositivos móviles y de escritorio.
- Definir y gestionar una base de datos relacional en MySQL, destinada a almacenar de forma eficiente y segura la información de usuarios, recetas y demás entidades necesarias para el funcionamiento de la aplicación.
- Incorporar un sistema de búsqueda y filtrado de recetas, que permita a los usuarios encontrar contenidos relevantes según nombre, ingredientes o categoría.
- Aplicar mecanismos de seguridad en el desarrollo de la aplicación, tales como validación de formularios, protección contra ataques comunes (como inyecciones SQL) y almacenamiento seguro de contraseñas.
- Aplicar buenas prácticas de desarrollo web tanto en el frontend como en el backend, incluyendo el uso de control de versiones con Git, documentación del código y separación de responsabilidades entre componentes.
- Realizar pruebas funcionales y de usuario, con el objetivo de verificar el correcto funcionamiento de todas las funcionalidades implementadas y asegurar una buena experiencia de uso.

5. ALCANCE DEL PROYECTO

5.1. Requisitos funcionales y no funcionales

Requisitos funcionales

Los requisitos funcionales describen las funcionalidades que el sistema debe ofrecer al usuario final. Para esta aplicación web de recetas de cocina, se han identificado los siguientes:

- Registro de usuarios: La aplicación debe permitir a nuevos usuarios registrarse mediante un formulario con nombre, correo electrónico y contraseña.
- Inicio de sesión y cierre de sesión: Los usuarios registrados deben poder autenticarse para acceder a funcionalidades personalizadas, así como cerrar sesión de forma segura.
- Gestión de perfil de usuario: Cada usuario debe poder visualizar y editar la información de su perfil, incluyendo su nombre, foto (opcional) y contraseña.
- Seguir usuarios: cada usuario tiene la opción de seguir a otros usuarios y viceversa, para poder ver las recetas de estos en su página de inicio de forma prioritaria.
- Publicación de recetas: Los usuarios autenticados podrán crear recetas introduciendo título, descripción, ingredientes, pasos de preparación e imagen.
- Edición y eliminación de recetas: Los usuarios podrán editar o eliminar únicamente las recetas que hayan creado.
- Visualización de recetas: Cualquier visitante podrá visualizar todas las recetas disponibles, incluyendo su contenido completo y la información del autor.
- Buscador de recetas: La aplicación debe ofrecer un sistema de búsqueda por palabras clave (título, ingredientes, etc.).
- Filtrado de recetas: El usuario podrá filtrar recetas por categorías predefinidas (por ejemplo: entrantes, platos principales, postres...).
- Sistema de comentarios: Los usuarios autenticados podrán comentar en las recetas de otros usuarios.
- Marcado de recetas favoritas: Los usuarios podrán marcar recetas como favoritas para consultarlas posteriormente.

Requisitos no funcionales

Los requisitos no funcionales definen características de calidad y restricciones técnicas que debe cumplir el sistema:

- Seguridad: Las contraseñas de los usuarios deben almacenarse cifradas. Se deben validar las entradas del usuario para evitar ataques como inyección SQL. Se debe implementar protección contra ataques CSRF en los formularios.
- Usabilidad: La interfaz debe ser intuitiva, fácil de usar y accesible para usuarios sin conocimientos técnicos. Se debe garantizar una experiencia fluida tanto en dispositivos móviles como en ordenadores.
- Rendimiento: Las páginas deben cargarse en un tiempo razonable (<2 segundos en condiciones normales). El sistema debe ser capaz de manejar múltiples usuarios y recetas sin disminuir significativamente su rendimiento.
- Compatibilidad: La aplicación debe ser compatible con los navegadores web más utilizados (Chrome, Firefox, Edge, Safari).
- Mantenibilidad: El código debe estar estructurado de forma clara y modular, utilizando buenas prácticas de programación para facilitar su mantenimiento y ampliación.
- Escalabilidad: La arquitectura basada en frontend y backend separados debe permitir futuras ampliaciones (por ejemplo, integración con APIs, app móvil, etc.).
- Portabilidad: El proyecto debe poder instalarse y desplegarse fácilmente en otros entornos utilizando herramientas comunes como Docker o entornos LAMP/WAMP.

5.2. Prototipado

Wireframing

Este es el boceto inicial de la aplicación (realizado con Miro), con el esquema y la funcionalidad que se pretende llevar a cabo:

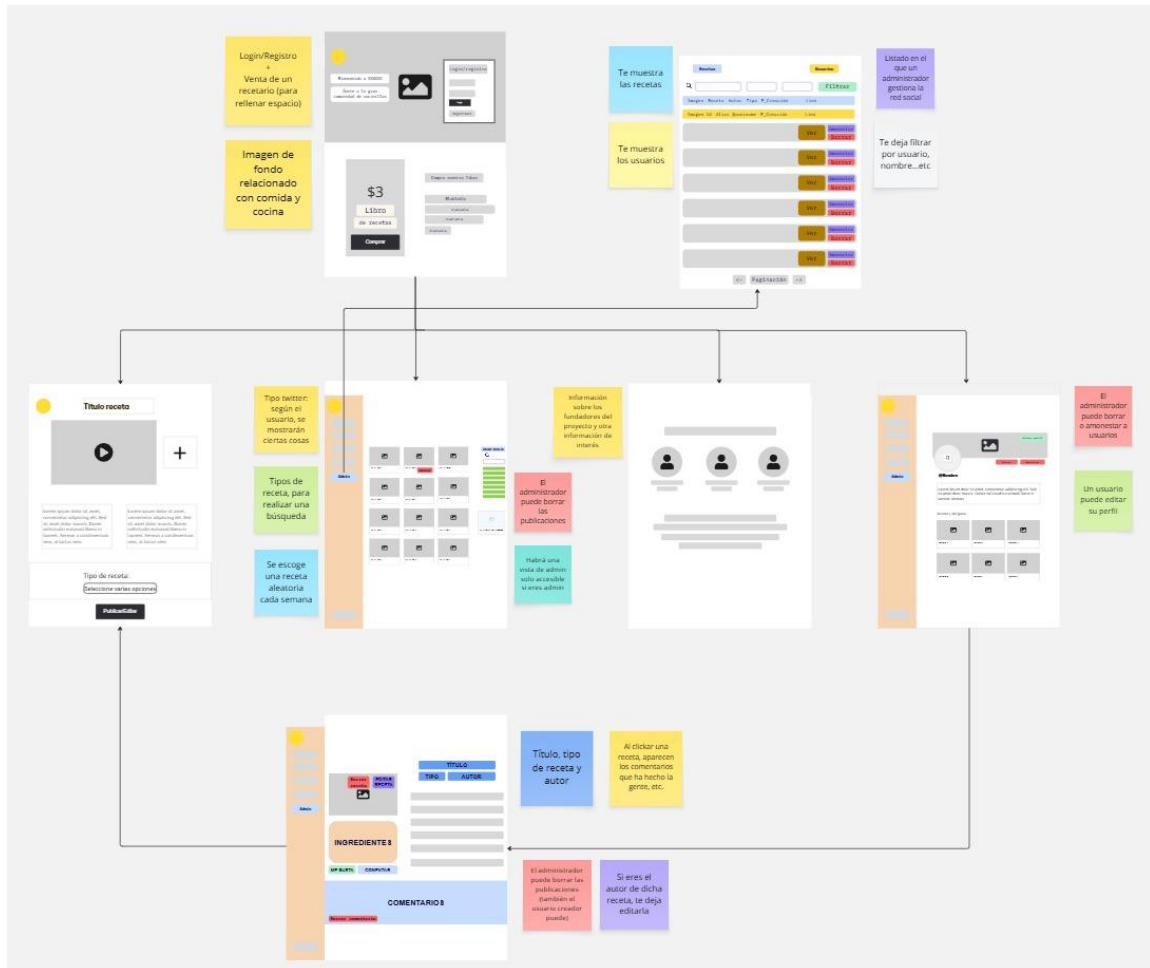


Ilustración 1 Wireframing

Bocetado de vistas

Aquí se muestra el bocetado del diseño inicial pretendido para cada vista de la aplicación. Destacar que con el desarrollo del proyecto se realizaron algunos cambios den dicho diseño, ya que se consideró que suponía una importante mejora.

- Landing page:

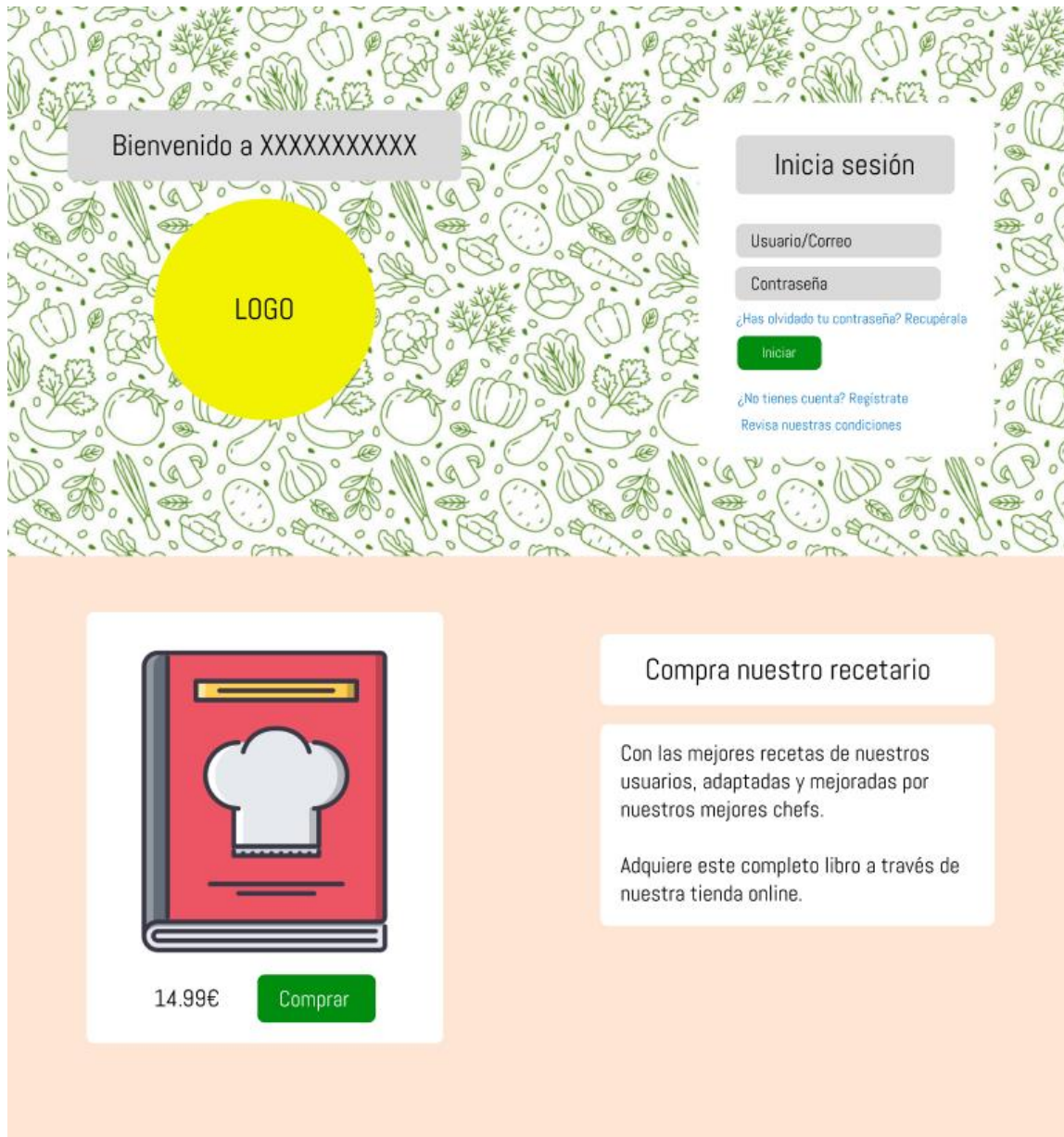


Ilustración 2 Vista "Landing page"

- Listado de recetas:

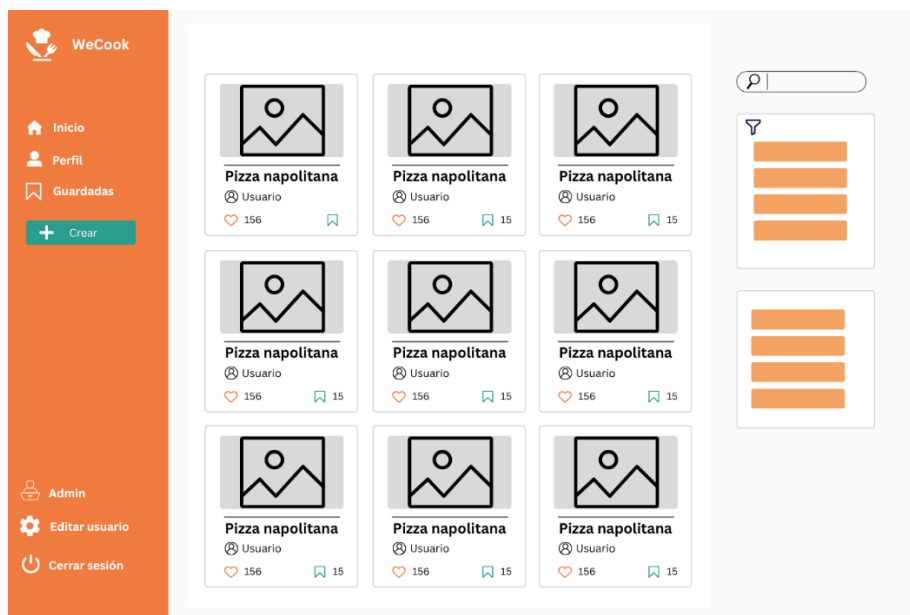


Ilustración 3 Vista "Listado de recetas"

- Perfil de usuario:

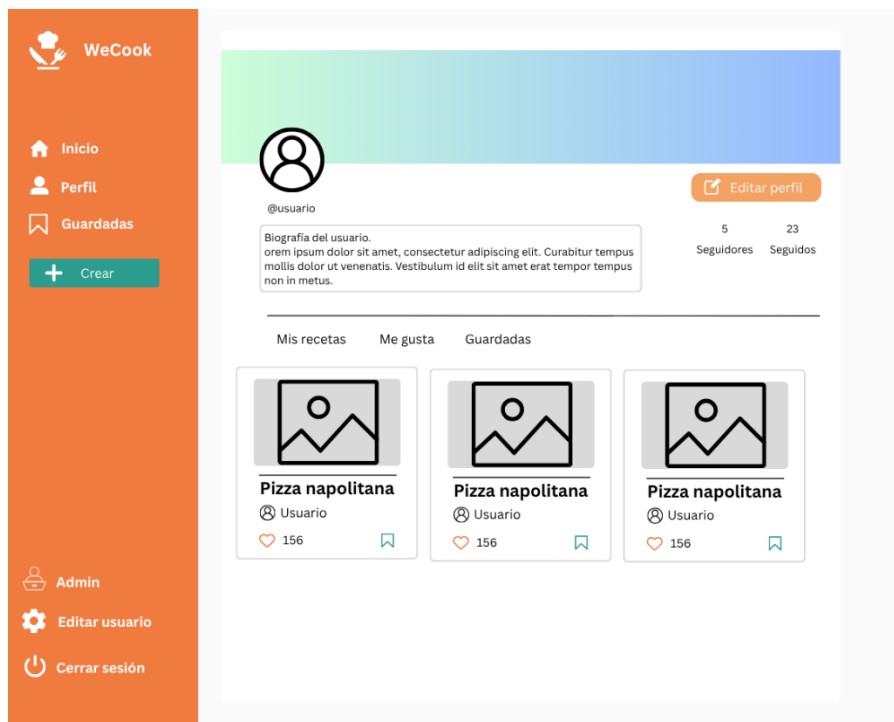


Ilustración 4 Vista "Perfil de usuario"

- Detalle de receta:

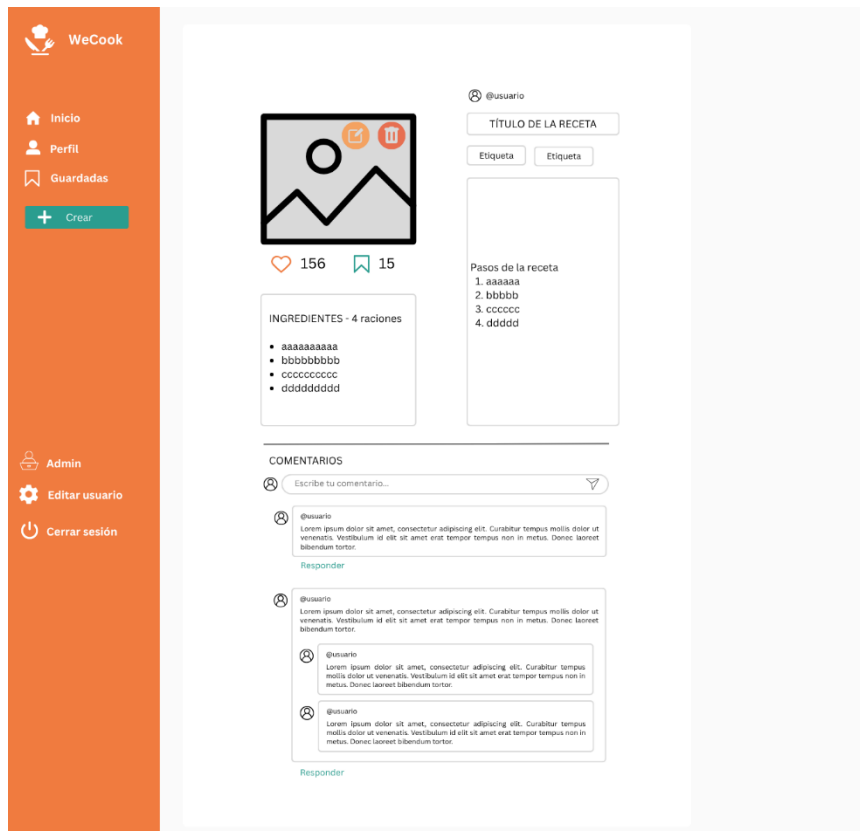


Ilustración 5 Vista "Detalle de receta"

- Recetas guardadas:

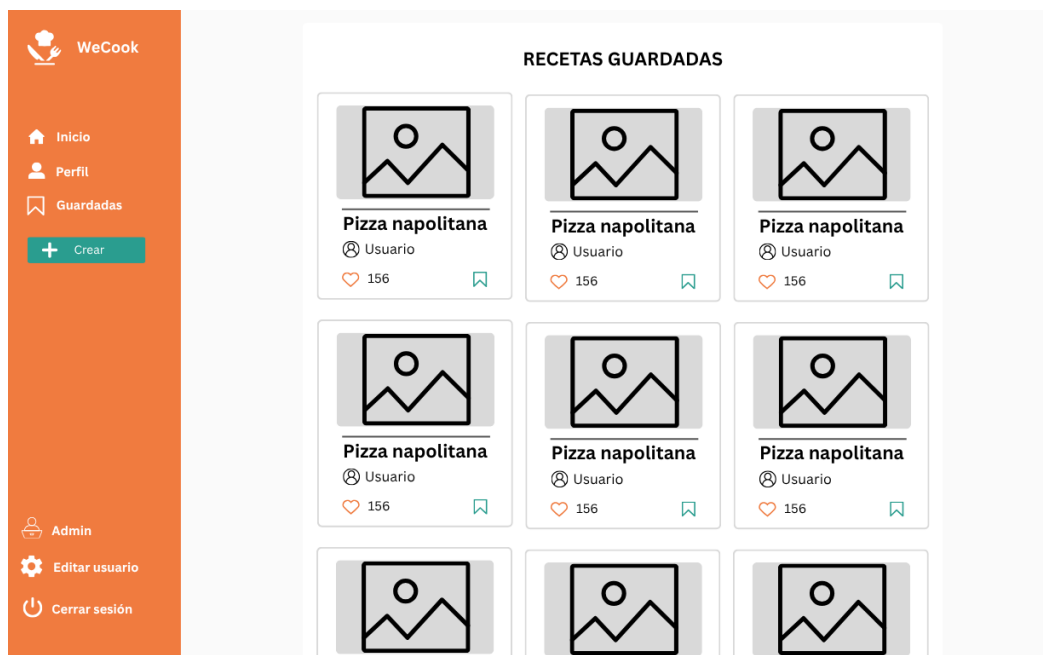


Ilustración 6 Vista "Recetas guardadas"

- Página de administrador:

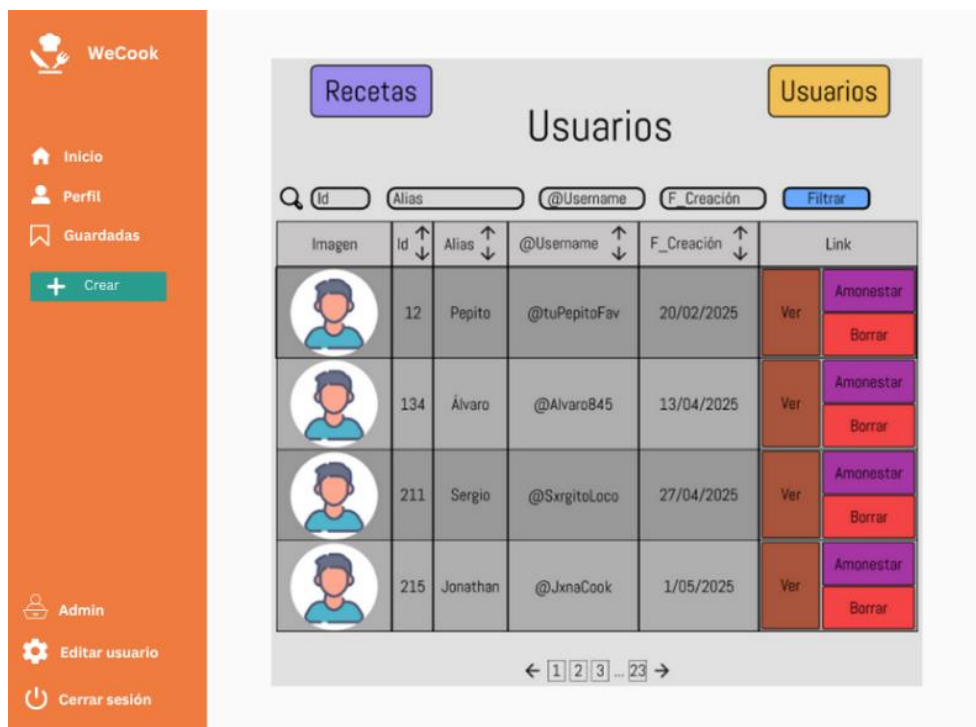


Ilustración 7 Vista "Administración"

- Página de editar cuenta:

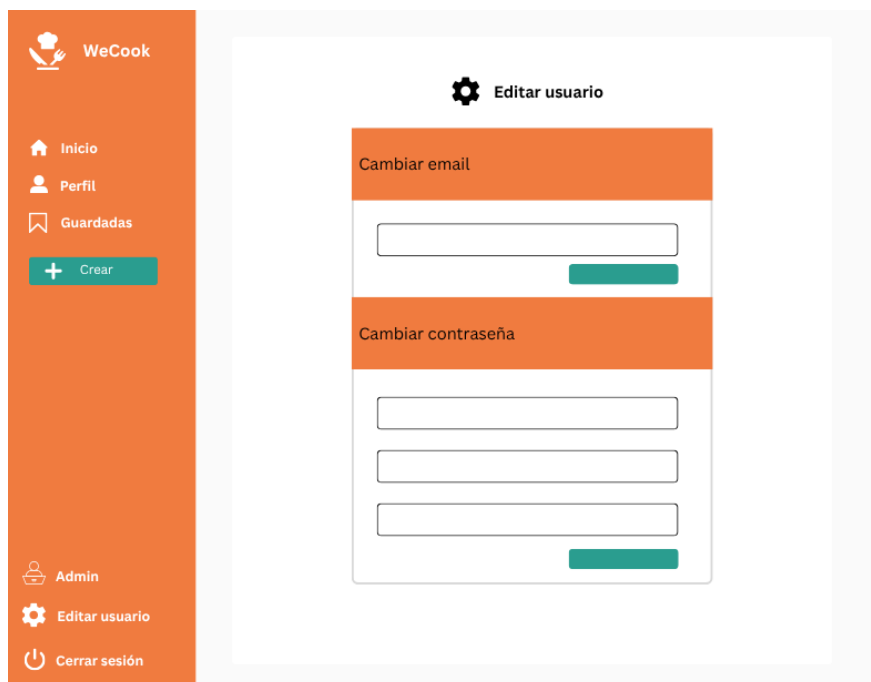


Ilustración 8 Vista "Editar cuenta"

- Página de contacto:

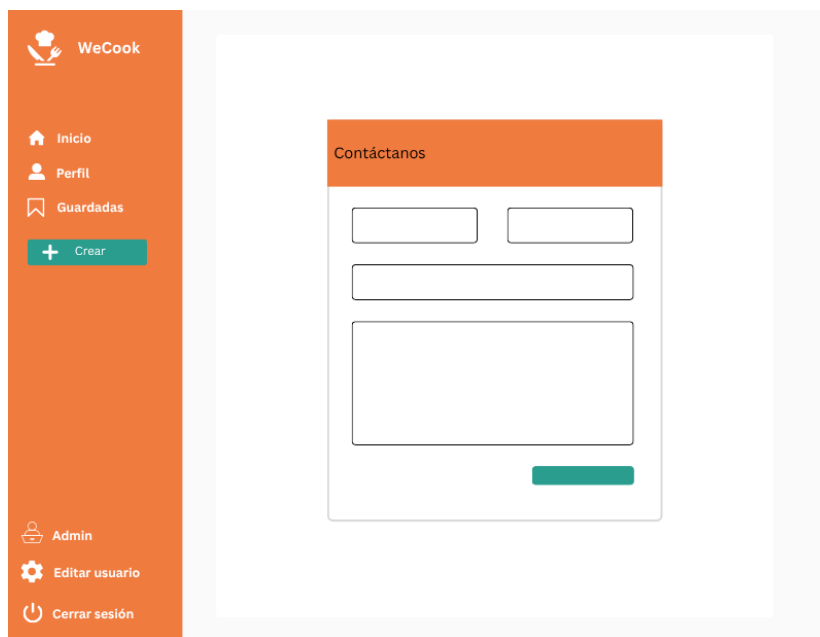


Ilustración 9 Vista "Contacto"

- Páginas de información:

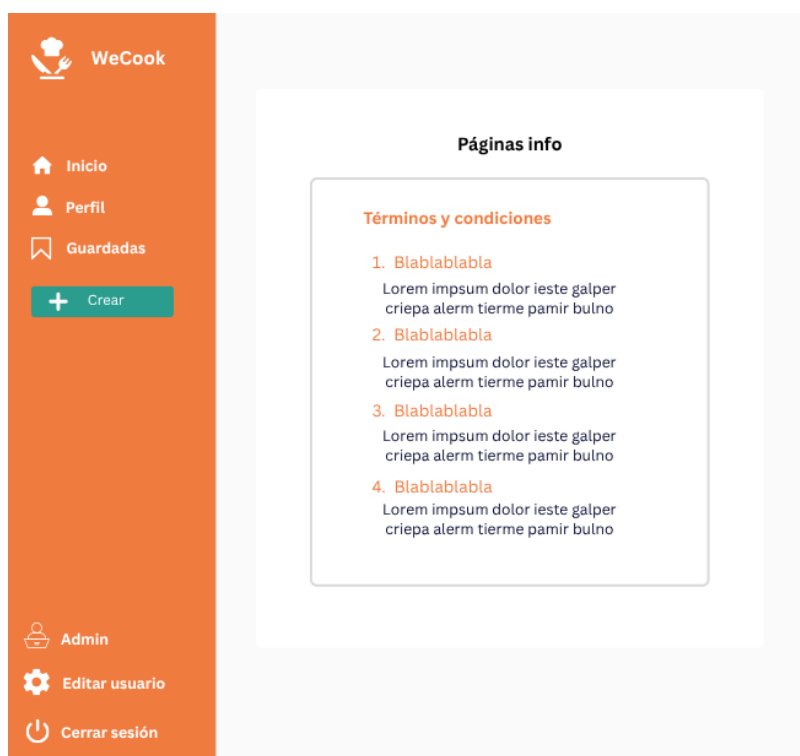


Ilustración 10 Vistas de información legal

Vistas en versión móvil

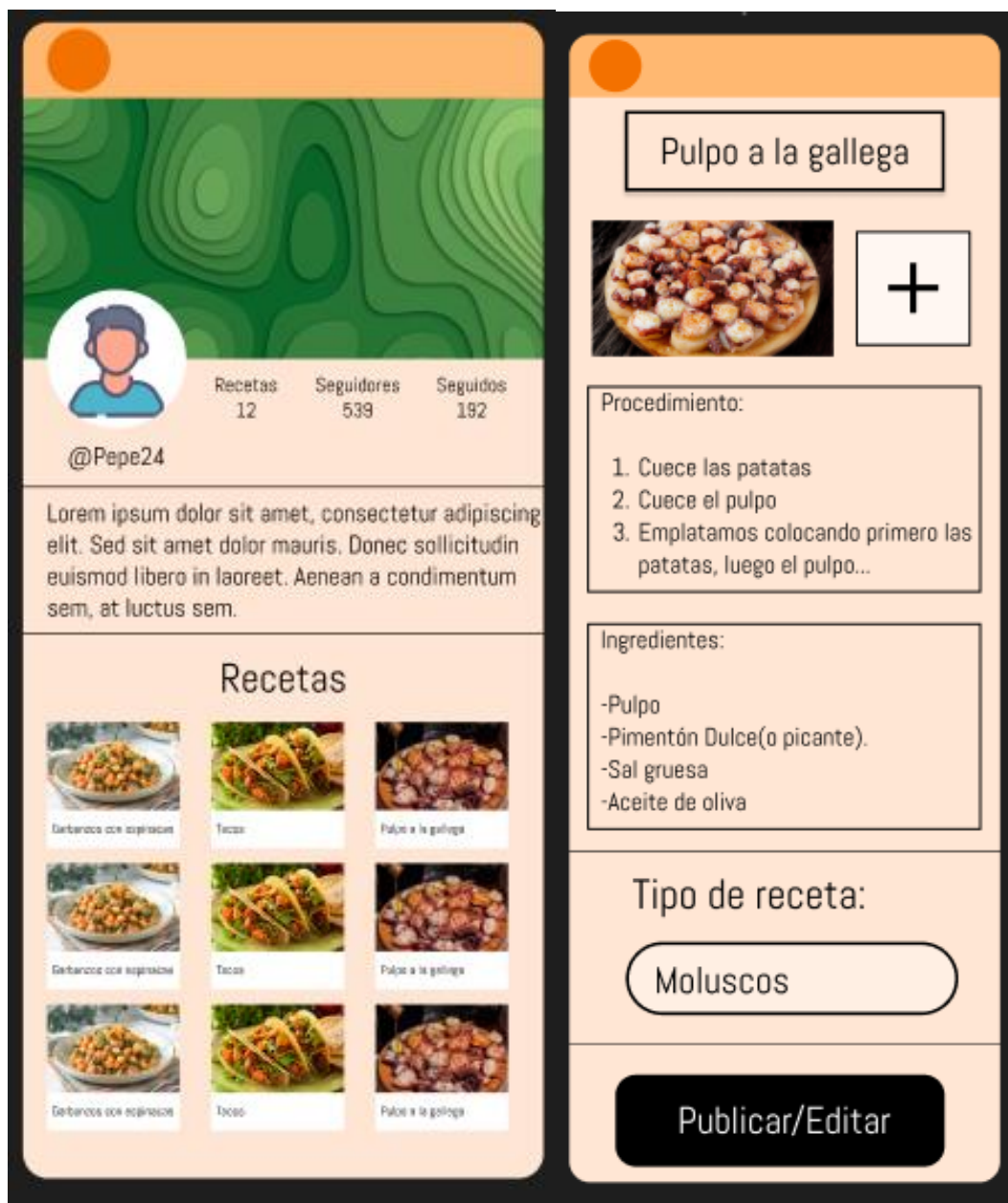


Ilustración 11 Vistas móvil "Perfil de usuario" y "Publicar/editar receta"

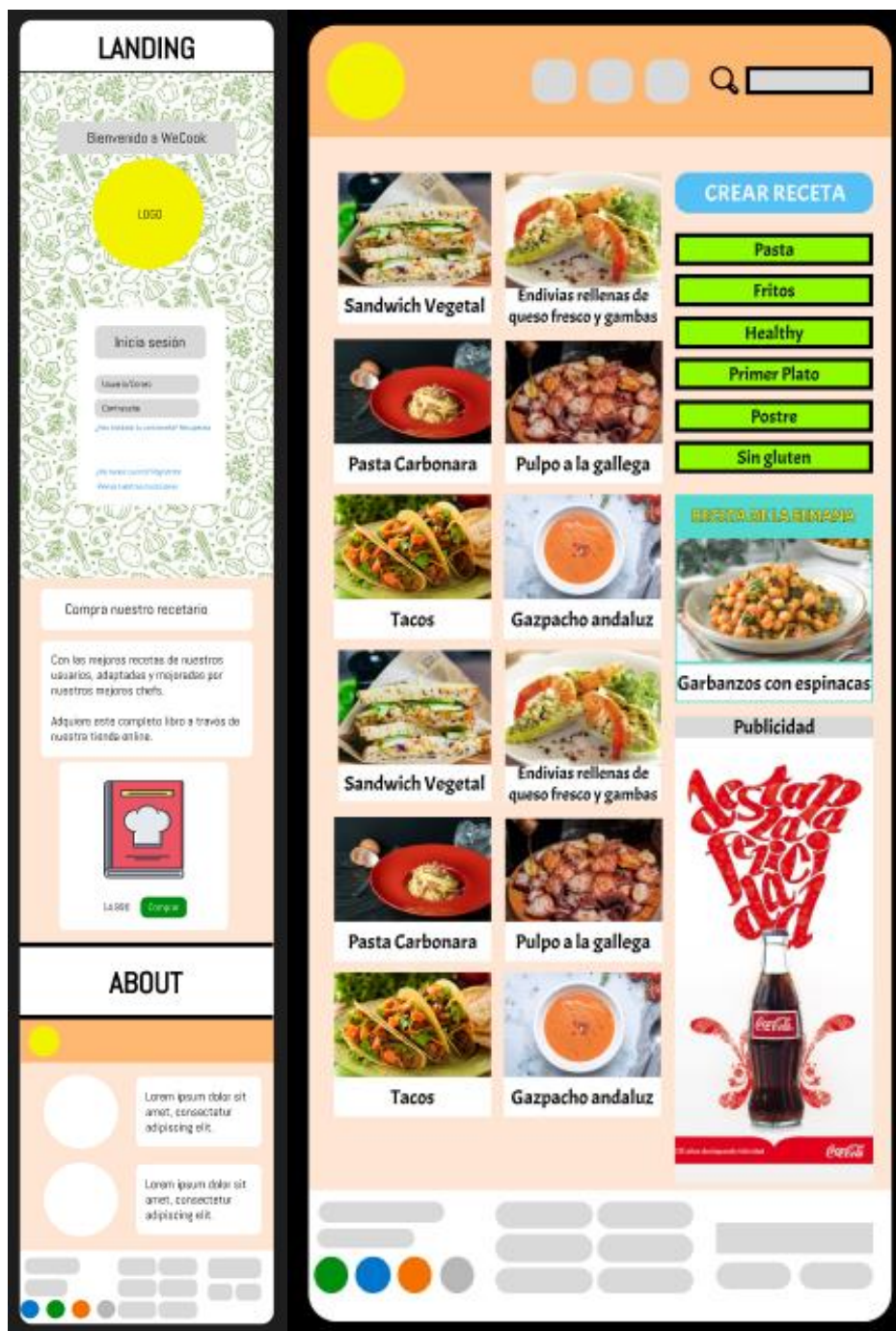


Ilustración 12 Vistas móvil "Landing page" y "Listado de recetas"



Ilustración 13 Vista móvil "Detalle de receta"

5.3. Tecnologías utilizadas

Laravel

Laravel es un framework de código abierto para desarrollar aplicaciones y servicios web con PHP. Su filosofía es desarrollar código PHP de forma elegante y simple, evitando el "código espagueti" empleando el patrón de arquitectura Modelo-Vista-Controlador (MVC), que facilita la organización del código separando la lógica de negocio, la interfaz y los datos.

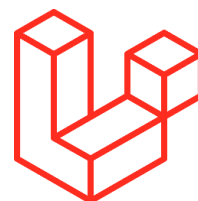


Ilustración 14 Logo de Laravel

Laravel ofrece utilidades que simplifican la creación de aplicaciones web, agilizando significativamente este proceso y generando un código organizado y manejable. Gracias a esta característica, la popularidad de Laravel ha experimentado un rápido crecimiento en los últimos tiempos, atrayendo a numerosos desarrolladores que lo eligen como su framework preferido para alcanzar una eficiente metodología de desarrollo.

Hemos utilizado esta tecnología debido a su popularidad y capacidad de manejar de forma sencilla algunas tareas complejas como la gestión de usuarios, seguridad y conexión a bases de datos. Además, está basado en PHP, lenguaje de programación con el cuál estamos familiarizados y nos ha facilitado tanto el proceso de aprendizaje como de desarrollo.

Blade

Blade es el motor de plantillas que viene integrado en Laravel y se utiliza para construir la parte visual de la aplicación. Permite escribir vistas de forma sencilla y estructurada, combinando HTML con una sintaxis específica que facilita la inclusión de lógica directamente en las plantillas, como bucles, condicionales y estructuras heredadas.



Ilustración 15 Logo de Blade

Una de las principales ventajas de Blade es que permite reutilizar componentes de la interfaz mediante plantillas y secciones, lo que promueve una mejor organización del código frontend. También permite el uso de directivas como `@if`, `@foreach`, `@include` o `@yield`, que simplifican mucho la creación de páginas dinámicas sin necesidad de mezclar directamente PHP con HTML.

Hemos elegido Blade como sistema de plantillas porque se integra perfectamente con Laravel y proporciona una forma clara, limpia y mantenible de construir las vistas. Su uso nos ha

permitido separar correctamente la lógica del backend y la presentación del frontend, favoreciendo así el desarrollo ágil y ordenado de la aplicación.

JavaScript

JavaScript es un lenguaje de programación interpretado, ampliamente utilizado en el desarrollo web para añadir interactividad y dinamismo a las páginas. Se ejecuta del lado del cliente, lo que permite modificar el contenido del HTML y CSS en tiempo real sin necesidad de recargar la página.



Ilustración 16 Logo de JavaScript

Esta tecnología es fundamental para nuestro proyecto ya que permite crear funcionalidades interactivas como validación de formularios, menús desplegables, animaciones y comunicación asíncrona con el servidor mediante AJAX. Gracias a su versatilidad y compatibilidad con todos los navegadores, JavaScript es esencial en cualquier aplicación web moderna.

Bootstrap

Bootstrap es un framework frontend que proporciona estilos CSS y componentes JavaScript listos para usar. Fue creado por Twitter y está enfocado en el diseño responsive, permitiendo que las aplicaciones se adapten automáticamente a diferentes tamaños de pantalla. Hemos utilizado Bootstrap para construir una interfaz visual atractiva y funcional sin necesidad de escribir mucho CSS personalizado. Gracias a sus componentes predefinidos como formularios, botones, menús y grillas, pudimos acelerar el desarrollo de la parte visual de nuestra aplicación y garantizar una buena experiencia en dispositivos móviles.

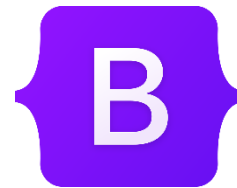


Ilustración 17 Logo de Bootstrap

MySQL

MySQL es un sistema de gestión de bases de datos relacional de código abierto. Permite almacenar, consultar y gestionar grandes cantidades de datos estructurados de manera eficiente mediante el uso de lenguaje SQL.



Ilustración 18 Logo de MySQL

Hemos utilizado MySQL como base de datos principal del proyecto, ya que permite definir relaciones entre entidades como usuarios, recetas o comentarios. Su compatibilidad con Laravel facilita el acceso a los datos de forma sencilla y segura.

MySQL Workbench

MySQL Workbench es una herramienta visual que permite modelar, administrar y consultar bases de datos MySQL de manera gráfica. Incluye funciones para diseñar esquemas, realizar consultas SQL, gestionar usuarios y analizar el rendimiento.



Ilustración 19 Logo MySQL Workbench

La hemos utilizado para diseñar y mantener la estructura de nuestra base de datos, visualizar relaciones entre tablas y ejecutar consultas de prueba, lo cual ha sido de gran ayuda en la fase de desarrollo y prueba del proyecto.

Git

Git es un sistema de control de versiones distribuido que permite gestionar el historial de cambios en el código fuente de un proyecto. Facilita la colaboración entre varios desarrolladores al permitir trabajar en ramas independientes y luego fusionar cambios de manera fácil y segura.



Ilustración 20 Logo de Git

En nuestro TFG, Git ha sido fundamental para trabajar en equipo sin conflictos, mantener un historial de versiones y poder recuperar estados anteriores del proyecto en caso de errores. Su integración con plataformas como GitHub lo convierte en una herramienta indispensable en cualquier entorno de desarrollo profesional.

GitHub

GitHub es una plataforma web para alojar repositorios de Git. Proporciona funcionalidades colaborativas y gestión de versiones, además de facilitar el trabajo en equipo y la revisión de código.



Ilustración 21 Logo de GitHub

Hemos utilizado GitHub para centralizar nuestro proyecto, compartir avances, y realizar control de versiones de forma remota. Su interfaz amigable y sus herramientas de colaboración han permitido una organización clara y efectiva del flujo de trabajo durante el desarrollo de la aplicación.

Visual Studio Code

Visual Studio Code (VS Code) es un editor de código fuente ligero, potente y multiplataforma desarrollado por Microsoft. Incluye herramientas de depuración, resaltado de sintaxis, autocompletado y un amplio ecosistema de extensiones.



Ilustración 22 Logo de VS Code

En nuestro proyecto, VS Code ha sido nuestro entorno de desarrollo principal, facilitando la escritura de código en Laravel, JavaScript y otros lenguajes gracias a su integración con Git, terminal integrada y compatibilidad con diversas extensiones específicas para cada tecnología utilizada.

Figma

Figma es una herramienta de diseño gráfico basada en la nube, especialmente orientada al diseño de interfaces de usuario (UI) y experiencia de usuario (UX). Permite trabajar de forma colaborativa en tiempo real, lo que facilita la creación de prototipos interactivos, wireframes y maquetas de alta fidelidad antes del desarrollo.



Ilustración 23 Logo de Figma

Hemos utilizado Figma para diseñar visualmente las pantallas de nuestra aplicación web, asegurándonos de que la experiencia de usuario sea intuitiva y atractiva. Gracias a sus funcionalidades de colaboración y su enfoque centrado en el diseño responsive, ha sido clave para definir la estructura visual del proyecto.

Miro

Miro es una plataforma colaborativa de pizarras digitales en línea diseñada para facilitar la planificación, organización y trabajo en equipo de forma visual. Permite a varios usuarios interactuar en tiempo real sobre una misma pizarra, añadiendo notas, diagramas, mapas mentales, esquemas y otros elementos visuales.



Ilustración 24 Logo de Miro

En nuestro TFG hemos utilizado Miro principalmente para planificar el proyecto, organizar ideas y definir estructuras de la aplicación como diagramas de flujo o esquemas de navegación. Su interfaz intuitiva y sus múltiples plantillas nos han permitido coordinar mejor el trabajo en grupo, clarificar tareas y tener una visión global del desarrollo.

Canva

Canva es una herramienta de diseño gráfico pensada para facilitar la creación de contenidos visuales de manera sencilla e intuitiva. Ofrece una amplia variedad de plantillas prediseñadas que permiten diseñar presentaciones, carteles y otros materiales gráficos.



Ilustración 25 Logo de Canva

En este trabajo se ha empleado Canva para elaborar elementos visuales de apoyo, como infografías y recursos gráficos (como el diseño de algunas vistas de la aplicación).

5.4. Diagrama de Entidad-Relación

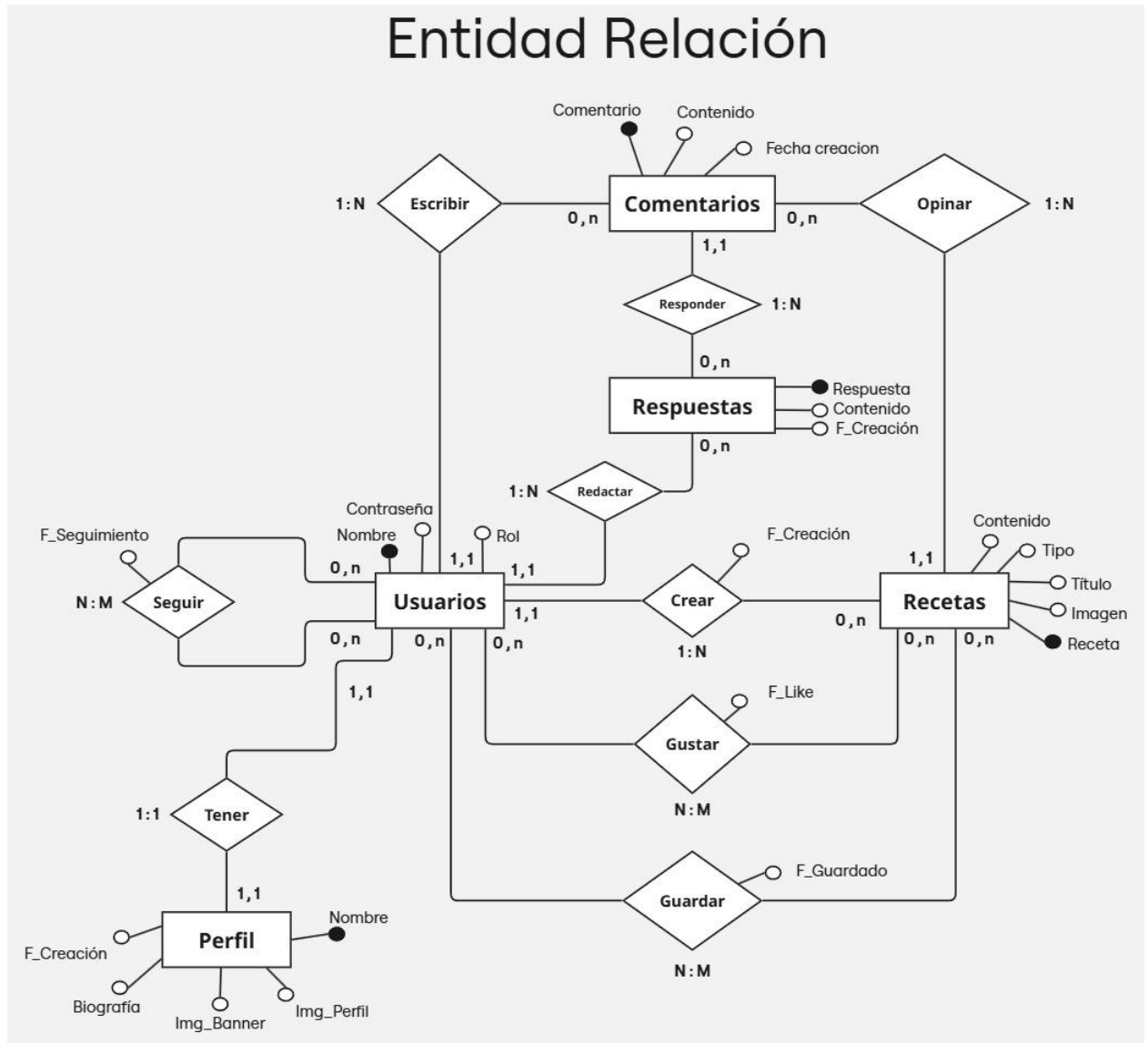


Ilustración 26 Diagrama entidad-relación

Base de datos

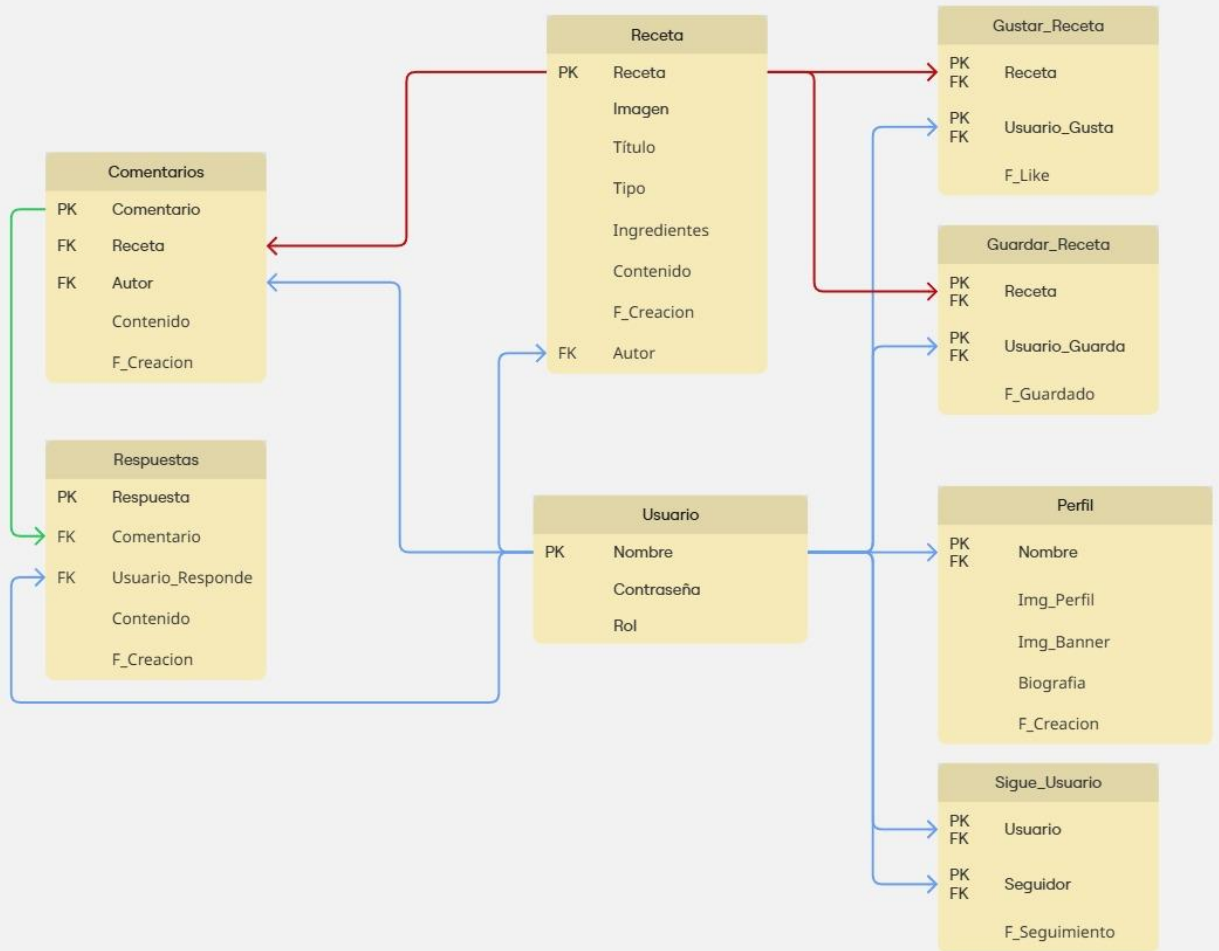


Ilustración 27 Esquema de la base de datos

5.5. Diagrama de casos de uso

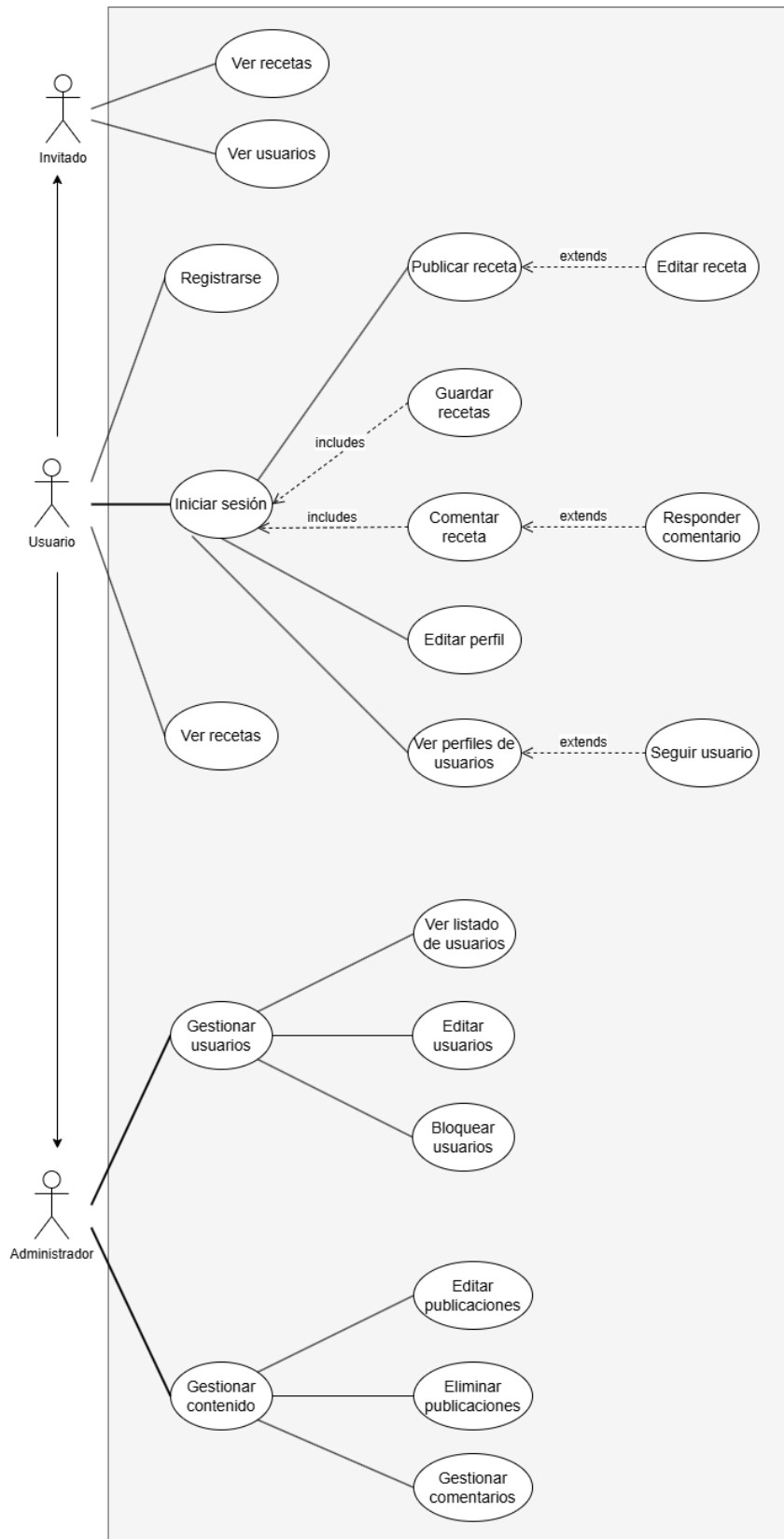


Ilustración 28 Diagrama de casos de uso

6. MARCO PRÁCTICO

Estructura

La estructura de carpetas y ficheros de este proyecto es la que trae Laravel en su versión 12 por defecto, como se muestra y se explica a continuación:

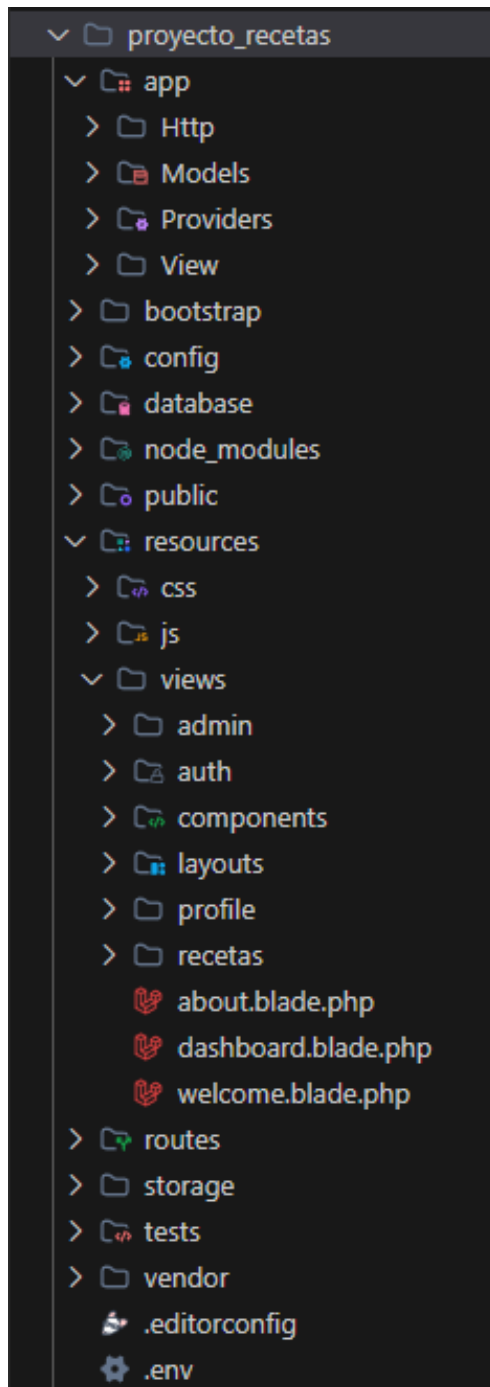


Ilustración 29 Estructura de carpetas de la aplicación

- Raíz del Proyecto (/proyecto_recetas)

Aquí se encuentran las carpetas y otros archivos importantes como:

- .env → Configuración del entorno (base de datos, mail, claves API...).
- artisan → Script de consola para ejecutar comandos Laravel.
- composer.json → Gestión de dependencias PHP.
- package.json → Dependencias y scripts del frontend.

- app/

Contiene el núcleo de la aplicación (la lógica de negocio). Aquí se agrupan:

- Http/Controllers/ → Controladores de la lógica web (patrón MVC).
- Models/ → Modelos que representan entidades y gestionan los datos de la base de datos.
- Http/Middleware/ → Filtros que se ejecutan al procesar una petición HTTP.
- Providers/ → Servicios que configuran e inyectan funcionalidades en la aplicación.

- bootstrap/

Contiene el archivo app.php, que se ejecuta al iniciar Laravel. Se usa para cargar y configurar el framework. También incluye la carpeta cache/ para el almacenamiento de archivos cacheados que mejoran el rendimiento.

- config/

Contiene archivos .php de configuración para todos los componentes de Laravel: base de datos (database.php), correo (mail.php), sesiones (session.php), etc. Permite modificar fácilmente el comportamiento de la aplicación.

- database/

Contiene todo lo relacionado con la base de datos:

- migrations/ → Archivos para crear/modificar la estructura de la base de datos.

- seeders/ → Archivos que insertan datos de prueba o iniciales.
- factories/ → Generadores de datos de prueba para testing.
- public/

Carpeta accesible públicamente desde el navegador. Contiene:

 - index.php → Punto de entrada a la aplicación.
 - Archivos públicos como imágenes, hojas de estilo (CSS), scripts (JS), etc.
- resources/

Contiene los archivos "front-end" de la aplicación:

 - views/ → Archivos Blade (.blade.php) que definen la interfaz del usuario.
 - lang/ → Archivos de traducción para aplicaciones multilinguaje.
 - css/y js/ → Archivos fuente que serán compilados para el frontend.
- routes/

Define las rutas de la aplicación. Laravel organiza las rutas por contexto:

 - web.php → Rutas para la web (HTML, vistas Blade).
 - api.php → Rutas para APIs REST.
 - console.php → Comandos de consola (Artisan).
 - channels.php → Rutas para canales de broadcasting (WebSockets).
- storage/

Almacena archivos generados o utilizados por la aplicación:

 - logs/ → Archivos de registro de errores.
 - app/ → Archivos del usuario (como subidas).
 - framework/ → Archivos temporales de cache, sesiones, vistas compiladas, etc.

Esta carpeta no debe ser accesible desde el navegador, excepto el subdirectorio `storage/app/public` si se enlaza correctamente ejecutando el comando `php artisan storage:link`.

- `tests/`

Contiene los tests automatizados:

- `Feature/` → Pruebas de alto nivel de funcionalidades completas.
- `Unit/` → Pruebas unitarias de clases específicas.

Laravel usa PHPUnit por defecto, pero puede integrarse con Pest u otros frameworks.

Landing page con inicio de sesión y registro de usuario



Ilustración 30 Sección principal de la landing page

La pantalla principal, donde se sitúa la ruta raíz, es la *landing page*. En ella se encuentra el inicio de sesión y el registro de usuario. Bajo esto, se encuentra una sección que muestra información sobre los desarrolladores de la aplicación (típica sección de información “Sobre nosotros” común en muchas webs) y otra pequeña sección sobre noticias culinarias destacadas.

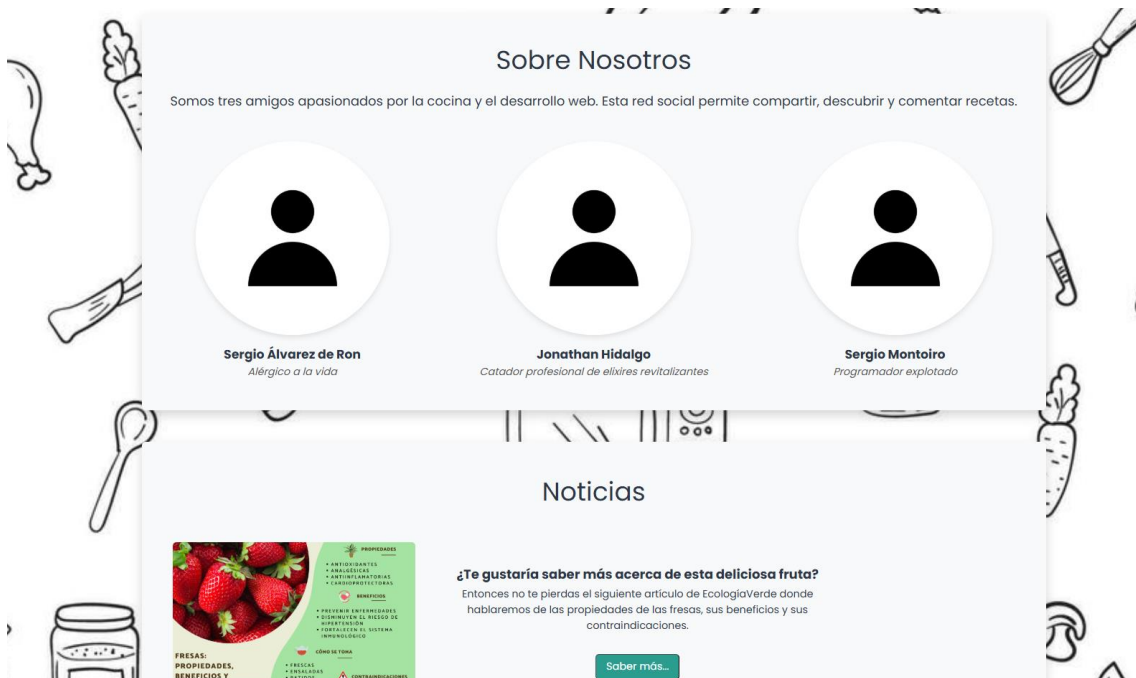


Ilustración 31 Secciones secundarias de la landing page

Aquí se define la ruta que lleva a esta página:

```
// Ruta raíz
Route::get('/', function () {
    return Auth::check() ? redirect()->route('recetas.lista') : view('welcome');
})->name('home');
```

Ilustración 32 Ruta raíz

Además, controla si el usuario ya está autenticado (es decir, ya ha iniciado sesión) y en tal caso, lo redirige a la página de listado de recetas a través de una llamada a uno de los controladores (la página predeterminada para los usuarios que ya iniciaron sesión). De esta forma se evitan problemas como cuando una sesión está iniciada, no pueda volver a iniciarse teniendo así múltiples sesiones simultáneas iguales abiertas.

Por otro lado, tenemos control de inicio erróneo de sesión. En caso de que un usuario trate de iniciar con un usuario inexistente o unas credenciales erróneas, se le redirige directamente a la pestaña de registro con un mensaje de error:

Bienvenido a WeCook

[Iniciar Sesión](#)

[Registrarse](#)

Únete a WeCook

Correo electrónico

pepe@email.com

auth.failed

Contraseña

Confirmar contraseña

Crear cuenta

Ilustración 33 Formulario de registro con error de validación de correo electrónico

En cuanto al registro de usuario, controla también con advertencias si el usuario que intentas registrar ya existe o si la repetición de contraseña es incorrecta:

Bienvenido a WeCook

[Iniciar Sesión](#)

[Registrarse](#)

Únete a WeCook

Correo electrónico

pepe@email.com

validation.unique

Contraseña

validation.confirmed

Confirmar contraseña

Crear cuenta

Ilustración 34 Formulario de registro con error de validación de contraseña

Si el usuario consigue registrarse correctamente, se le redirige a la vista de listado de recetas (ruta predeterminada para usuarios autenticados).

```
public function store(LoginRequest $request): RedirectResponse
{
    $request->authenticate();

    $request->session()->regenerate();

    return redirect()->route('recetas.lista');
    // return redirect()->intended(route('dashboard', absolute: false));
}
```

Ilustración 35 Función que maneja redirección de usuarios registrados

Todo esto lo manejan los controladores de autenticación por defecto que trae Laravel, facilitando mucho el trabajo. En uno de los controladores, hemos creado una función que crea al usuario de forma distinta a como viene por defecto, pues para cada usuario queremos crear un perfil al que se le asocia, que pueda editar a su gusto. En esta imagen se ve la función del controlador que crea el perfil del usuario según se registra.

```
class RegisteredUserController extends Controller
{
    public function store(Request $request): RedirectResponse
    {
        $request->validate([
            // 'name' => ['required', 'string', 'max:255'],
            'email' => ['required', 'string', 'lowercase', 'email', 'max:255', 'unique:'.User::class],
            'password' => ['required', 'confirmed', Rules::Password::defaults()],
        ]);

        $user = User::create([
            // 'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password),
            'user_type' => 0,
        ]);

        Perfil::create([
            'id user' => $user->id,
            'name' => $user->email, //Poner el nombre del usuario
            'img_perfil' => "images/default-profile.jpg",
            'img_banner' => "images/default-banner.jpg",
            'biografia' => "¡Cocinando en WeCook!",
        ]);

        event(new Registered($user));

        Auth::login($user);

        return redirect('recetas');
    }
}
```

Ilustración 36 Controlador de usuarios con función de registro de usuarios

Recordar contraseña:

Si el usuario necesita recuperar su contraseña, puede acceder al enlace que se sitúa justo debajo de la sección de inicio de sesión, que le llevará a esta vista:

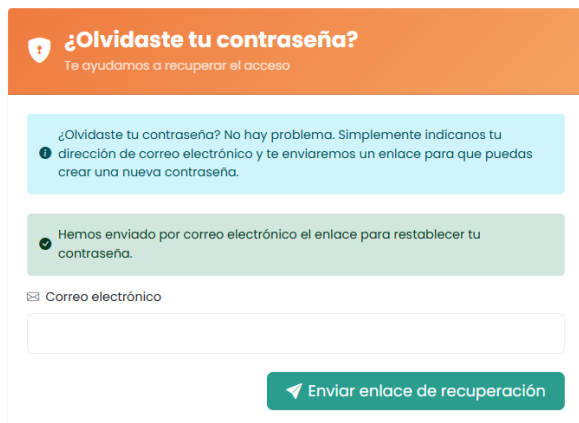


Ilustración 37 Formulario de recordar contraseña

Esto enviará un mensaje a su correo electrónico como el siguiente (y muestra un aviso de que se envió correctamente, como se ve en la anterior imagen en el recuadro verde):

Hello!

You are receiving this email because we received a password reset request for your account.

Reset Password

This password reset link will expire in 60 minutes.

If you did not request a password reset, no further action is required.

Regards,
Laravel

If you're having trouble clicking the "Reset Password" button, copy and paste the URL below into your web browser: <http://127.0.0.1:8000/reset-password/17dc97cf4628bd88965ba85079b0b794362142ae9f413cba37b1edd2e85ef942?email=pepe%40email.com>

Ilustración 38 Correo recibido para recuperar contraseña

Pulsar el botón del mensaje, le llevará a otra vista, donde podrá cambiar su contraseña por una nueva y de esta forma poder iniciar sesión con ella:



Restablecer contraseña
Crea una nueva contraseña segura

Correo electrónico
pepe@email.com

Contraseña

Confirmar contraseña

✓ Restablecer contraseña

Ilustración 39 Formulario de restablecer contraseña

El envío de correos electrónicos se gestiona desde los controladores por defecto de Laravel, que se encuentran dentro de la carpeta /controllers/auth. Por ejemplo: este controlador se encarga de enviar el enlace de reinicio de contraseña al usuario.

```
class PasswordResetLinkController extends Controller
{
    /**
     * Display the password reset link request view.
     */
    public function create(): View
    {
        return view('auth.forgot-password');
    }

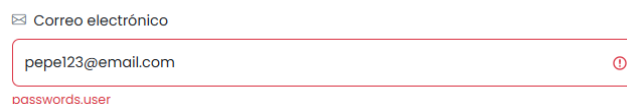
    /**
     * Handle an incoming password reset link request.
     *
     * @throws \Illuminate\Validation\ValidationException
     */
    public function store(Request $request): RedirectResponse
    {
        $request->validate([
            'email' => ['required', 'email'],
        ]);

        // We will send the password reset link to this user. Once we have attempted
        // to send the link, we will examine the response then see the message we
        // need to show to the user. Finally, we'll send out a proper response.
        $status = Password::sendResetLink(
            $request->only('email')
        );

        return $status == Password::RESET_LINK_SENT
            ? back()->with('status', __($status))
            : back()->withInput($request->only('email'))
                ->withErrors(['email' => __($status)]);
    }
}
```

Ilustración 40 Funciones que manejan la recuperación de contraseña

Maneja el correo específico que requiere la actualización de contraseña y así controla la seguridad, pues no permite acceder desde enlaces diferentes al cambio de contraseña de un correo concreto. Si se trata de poner otro correo, se mostrará un aviso y no se realizará la acción:



Correo electrónico

pepel23@email.com

passwords.user

Ilustración 41 Formulario con error de validación de correo electrónico

Además, una vez realizado el cambio de contraseña, el enlace pierde la efectividad y, en lugar de redirigir nuevamente a la vista de restablecer contraseña, lleva al usuario a la página principal.

Verificación de usuario

Cuando un usuario se registra, se le envía un correo electrónico con el cual se ha registrado, para realizar una autenticación de seguridad de que se registró con un correo correcto.

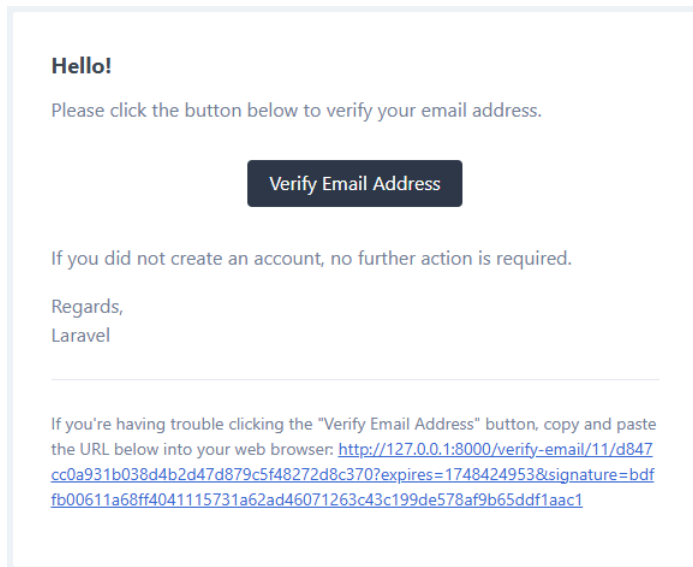


Ilustración 42 Correo recibido de verificación de usuario registrado

Esto es manejado por el siguiente controlador:

```
class VerifyEmailController extends Controller
{
    /**
     * Mark the authenticated user's email address as verified.
     */
    public function __invoke(EmailVerificationRequest $request): RedirectResponse
    {
        if ($request->user()->hasVerifiedEmail()) {
            return redirect()->intended(route('recetas.lista', absolute: false).'?verified=1');
        }

        if ($request->user()->markEmailAsVerified()) {
            event(new Verified($request->user()));
        }

        return redirect()->intended(route('recetas.lista', absolute: false).'?verified=1');
    }
}
```

Ilustración 43 Función que verifica y redirige al usuario

Este controlador también se genera por defecto por Laravel, aunque modificamos la ruta por defecto a la que el enlace del correo electrónico redirige tras autenticar correctamente al usuario (en nuestro caso, decidimos redirigir al listado de recetas).

Barra de navegación

Para todas las vistas de los usuarios autenticados (es decir, aquellos que han iniciado sesión), se muestra una barra de navegación (la cual se puede minimizar y ampliar), con diversos enlaces al resto de vistas y funcionalidades de la aplicación.

Tanto la barra de navegación como las diferentes vistas se renderizan en el siguiente archivo `app.blade.php`:

```
<body class="font-sans antialiased">
  <div class="d-flex min-vh-100 flex-column flex-lg-row position-relative"> <!-- Añadi
    @auth
      <!-- Sidebar de móvil y tablet -->
      <aside id="sidebar-responsive">
        @include('layouts.navigationResponsive')
      </aside>

      <!-- Sidebar de escritorio (lg en adelante) -->
      <aside id="sidebar-desktop">
        @include('layouts.navigation')
      </aside>
    @endauth

    <!-- Modal Crear Receta -->
    @include('modals.crear-receta')

    {{-- @guest
      @include('layouts.navigationGuest')
    @endguest --}}

    <!-- Contenido principal -->
    <main id="mainContent" class="main-content bg-light d-flex justify-content-cen
      <div class="container-fluid rounded-3 shadow-sm p-3 bg-white" style="max-w
        @yield('content')
      </div>
    </main>

    <!-- Footer fuera del contenedor principal para evitar problemas de z-index -->
    @include('layouts.footer')
  </div>
  @yield('js')
  @stack('scripts')
</body>
```

Ilustración 44 Plantilla que maneja las vistas y secciones de la aplicación

Listado de recetas:

Esta es la vista principal predeterminada para los usuarios con sesión iniciada. Aquí se muestran las recetas publicadas por los usuarios. Permite visualizar las recetas en detalle pulsando sobre ellas, además de dar me gusta y guardar recetas con los botones y acceder al perfil del usuario creador de cada receta pulsando sobre su nombre.

Por otro lado, se pueden realizar búsquedas de recetas por filtros y búsqueda de usuarios.

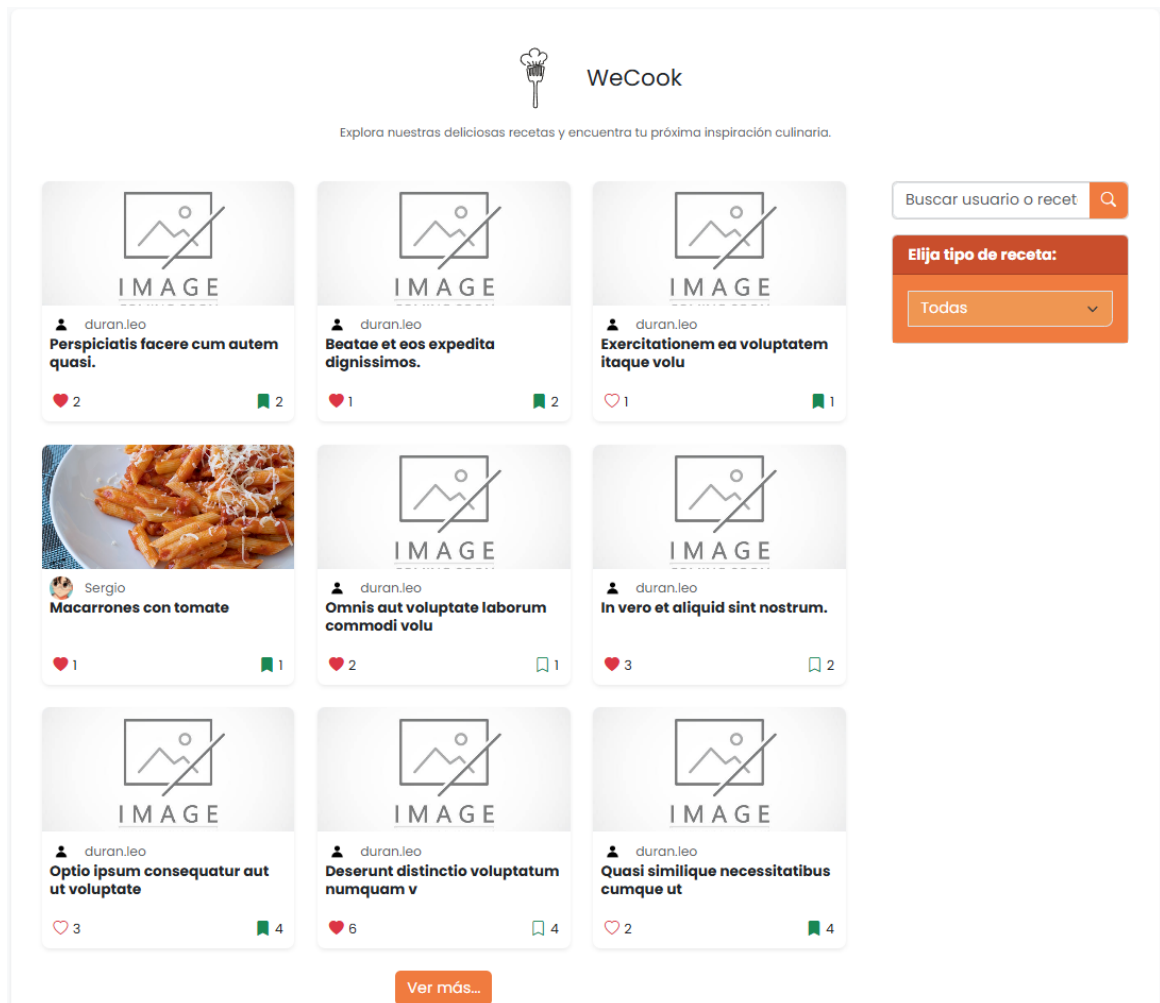


Ilustración 45 Listado de recetas

Todo esto se gestiona a través de las funciones del controlador específico para manejar las recetas:

```

class RecetaController extends Controller {
    public function listarRecetasPrincipalAjax(Request $request){

        $listaIds = [];
        $filtro = $request->tipo;

        if(isset($request->recetas)){

            $listaIds = array($request->recetas);

            for($i = 0; $i<count($listaIds[0]);$i++){
                $listaIds[0][$i] = intval($listaIds[0][$i]);
            }

            if($filtro == "Todas"){

                $recetas = Receta::whereIn('id',$listaIds[0])->whereNot('autor_receta',Auth::id())->get();
            }else{

                $recetas = Receta::whereIn('id',$listaIds[0])->where('tipo',$filtro)->whereNot('autor_receta',Auth::id())->get();
            }
        }else{

            if($filtro == "Todas"){

                $recetas = Receta::orderBy('created_at', 'desc')->whereNot('autor_receta',Auth::id())->get();
            }else{

                $recetas = Receta::orderBy('created_at', 'desc')->where('tipo',$filtro)->whereNot('autor_receta',Auth::id())->get();
            }
        }

        $recetas = $recetas->shuffle()->take(9);

        foreach($recetas as $r){

            $r['meGustas'] = count($r->usuariosQueGustaron);
            $r['vecesGuardados'] = count($r->usuariosQueGuardaron);
            $r['nombreAutor'] = $r->autor->perfil->name;
            $r['like'] = GustarReceta::where('id_receta',$r->id)->where('id_user',Auth::id())->exists();
            $r['guardado'] = GuardarReceta::where('id_receta',$r->id)->where('id_user',Auth::id())->exists();
        }
    }
}

```

Ilustración 46 Controlador de recetas con función que muestra el listado de recetas

Esta función concreta del controlador lista las recetas mediante AJAX, lo cual permite la actualización dinámica de las mismas sin necesidad de recargar la página, permitiendo una navegación mucho más cómoda y fluida para el usuario.

Las recetas que se muestran dependen de los usuarios a los que sigues y de la popularidad de las recetas en base a la cantidad de “Me gusta” que reciben. De esta forma, este simple algoritmo muestra mayor diversidad de recetas que si simplemente se muestran por fecha de creación.

Detalle de receta

Esta vista muestra la información concreta de la receta, así como todos los comentarios y respuestas de los usuarios.

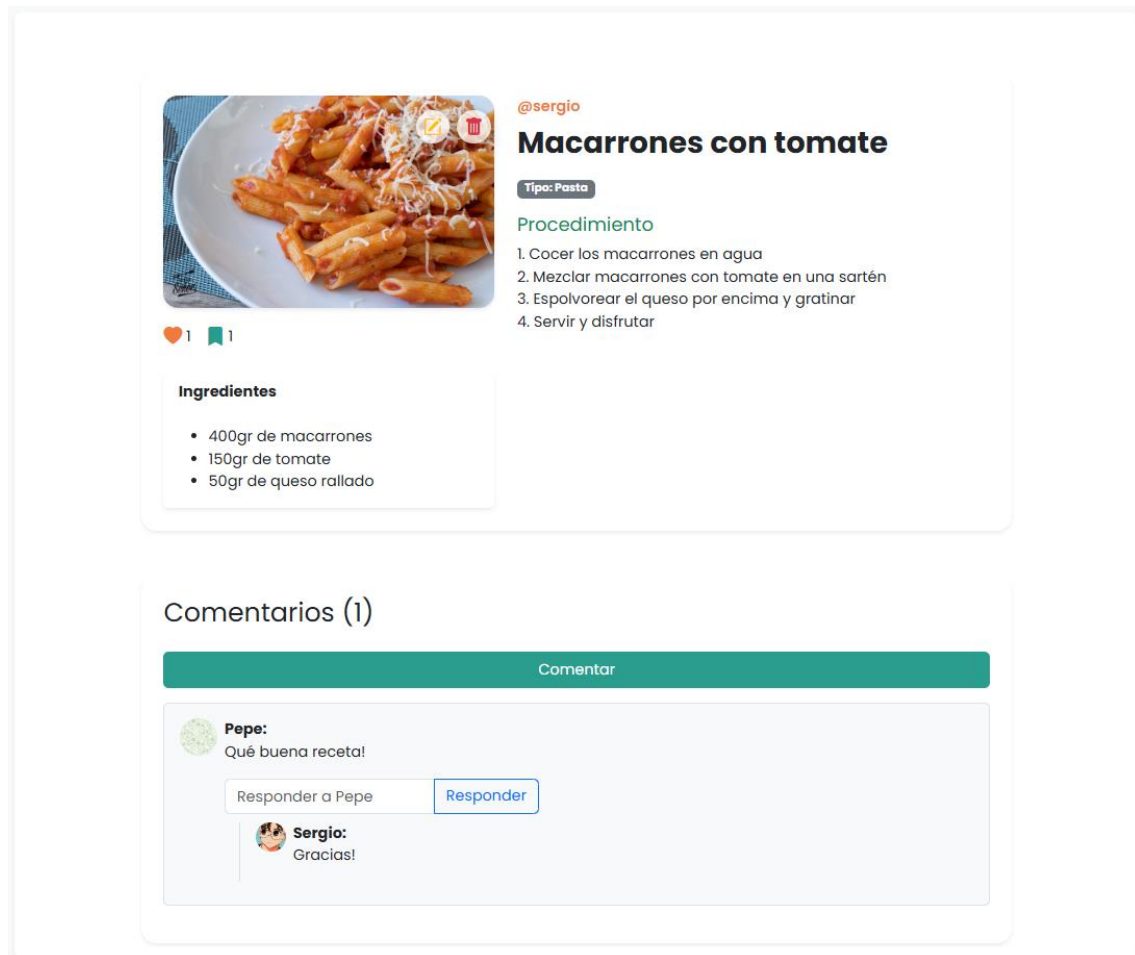


Ilustración 47 Detalle de receta y comentarios y respuestas de usuarios

Esta función del controlador se encarga de manejar esta vista en función de cada receta (según su ID). En función del usuario que visualice la receta, tendrá diferentes opciones disponibles. Si es el usuario creador, se le mostrarán botones para editar y eliminar la receta, pero no podrá dar me gusta ni guardar su propia receta. Los usuarios con sesión iniciada podrán dar me gusta, guardar la receta y comentar. Por último, los usuarios invitados (es decir, que no han iniciado sesión) tan solo podrán visualizar dicha receta, sin poder realizar ninguna otra acción.

En este fragmento de código se muestran las funciones del controlador que permitan al propietario de la receta editar a esta misma:

```
// Mostrar formulario con datos actuales
public function editarReceta($id) {
    $receta = Receta::findOrFail($id);
    if (Auth::id() !== $receta->autor_receta) {
        abort(403, 'No autorizado.');
```

Ilustración 48 Funciones de editar y actualizar receta

Una de ellas redirige al modal de edición con los datos de dicha receta escritos desde el inicio para editarlos. La otra, se encarga de persistir la información en la base de datos (realizando las comprobaciones y validaciones pertinentes) y redirige de nuevo al listado tras la actualización.

Por otro lado, se muestra también la función que permite al usuario eliminar su propia receta:

```
// Eliminar receta
public function eliminarReceta($id) {
    $receta = Receta::findOrFail($id);
    if (Auth::id() !== $receta->autor_receta) {
        abort(403, 'No autorizado.');
```

Ilustración 49 Función de eliminar receta

Realiza las comprobaciones necesarias para impedir que otros usuarios eliminen recetas que no son suyas y elimina la receta de la base de datos si la acción se realiza con éxito.

Además, se muestran avisos con doble verificación para eliminar las recetas para mayor seguridad e impedir el borrado de información de forma indeseada:

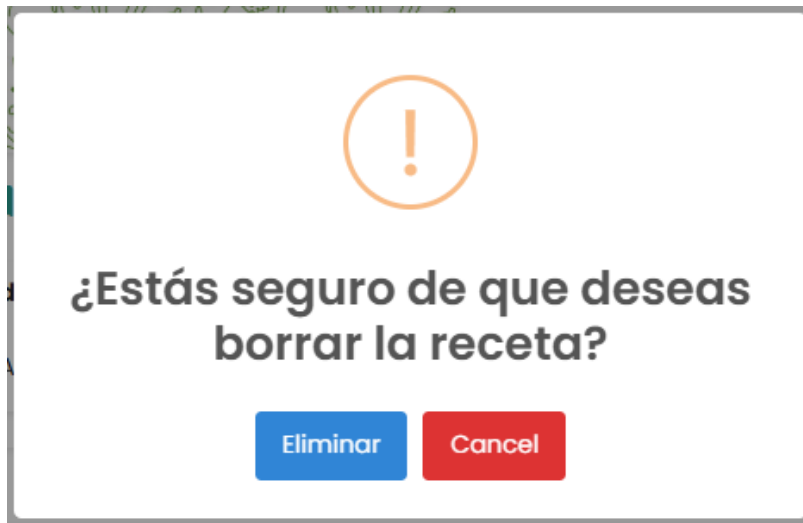


Ilustración 50 Pop-up de doble confirmación para eliminar receta

También se muestra otra alerta avisando si la acción de borrado se ha realizado correctamente:

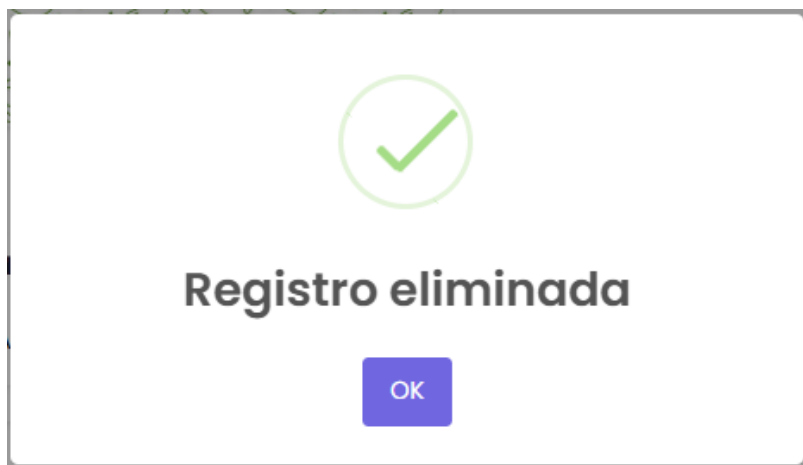


Ilustración 51 Pop-up con confirmación de receta eliminada

En cuanto a las funcionalidades de gustar y guardar las recetas, se manejan desde las siguientes funciones del mismo controlador:

```

public function guardarRecetaUsuarioAjax($id)
{
    $userId = Auth::id();

    // Verificar si ya está guardada
    $yaExiste = GuardarReceta::where('id_receta', $id)->where('id_user', $userId)->exists();

    if (!$yaExiste) {
        GuardarReceta::create([
            'id_receta' => $id,
            'id_user' => $userId,
            'f_guardar' => now(),
        ]);

        return response()->json(['status' => 'success', 'message' => 'Guardada']);
    }

    return response()->json(['status' => 'failed', 'message' => 'Ha ocurrido un error']);
}

```

Ilustración 52 Función de guardado de receta

Comprueba si el usuario ha iniciado sesión para permitirle realizar la acción y en caso afirmativo, verifica si la receta ya está guardada o no. En caso de no estarlo aún, realiza el guardado de la receta y lo persiste en base de datos haciendo uso del modelo.

```

public function eliminarGuardadoAjax($id)
{
    $receta = GuardarReceta::where('id_receta', $id)->where('id_user', Auth::id());

    if($receta){
        $receta->delete();
        return response()->json(['status' => 'success', 'message' => 'Se ha eliminado la receta']);
    }

    return response()->json(['status' => 'failed', 'message' => 'Ha ocurrido un error']);
}

```

Ilustración 53 Función de eliminar receta

Esta misma situación se replica de manera similar, pero para eliminar las recetas de los guardados del usuario.

Por otro lado, tenemos el mismo funcionamiento para las recetas gustadas:

```

public function gustarRecetaUsuarioAjax($id)
{
    $userId = Auth::id();

    // Verificar si ya le dio me gusta
    $yaExiste = GustarReceta::where('id_receta', $id)->where('id_user', $userId)->exists();

    if (!$yaExiste) {
        GustarReceta::create([
            'id_receta' => $id,
            'id_user' => $userId,
            'f_gustar' => now(),
        ]);

        return response()->json(['status' => 'success', 'message' => 'Dado me gusta']);
    }

    return response()->json(['status' => 'failed', 'message' => 'Ha ocurrido un error']);
}

```

Ilustración 54 Función para marcar recetas gustadas

```

public function eliminarMeGustaAjax($id)
{
    $receta = GustarReceta::where('id_receta', $id)->where('id_user', Auth::id());

    if($receta){
        $receta->delete();
        return response()->json(['status' => 'success', 'message' => 'Se ha eliminado el me gusta']);
    }

    return response()->json(['status' => 'failed', 'message' => 'Ha ocurrido un error']);
}

```

Ilustración 55 Función para eliminar marca de recetas gustadas

Cabe destacar que en ambos casos (para guardar o dar me gusta a recetas), las acciones se realizan a través de AJAX para facilitar la navegación, pues no es necesario que la página se recargue completamente, si no que solo se actualizan aquellas partes de la vista que se ven afectadas por estas acciones.

Como último apartado a mencionar sobre esta vista del detalle de recetas, mencionar el apartado de los comentarios y respuestas. Este apartado se maneja desde controladores diferentes, como se muestra a continuación:

```

class ComentarioController extends Controller {

    public function store(Request $request) {
        $request->validate([
            'contenido' => 'required|string|max:1000',
            'id_receta' => 'required|exists:recetas,id',
        ]);

        Comentario::create([
            'id_user' => Auth::id(),
            'id_receta' => $request->id_receta,
            'contenido' => $request->contenido,
            'f_creacion' => now(),
        ]);

        return back()->with('success', 'Comentario publicado correctamente.');
```

Ilustración 56 Función para comentar receta

Este fragmento de código se encarga de realizar las validaciones y persistencia en base de datos que corresponde a los comentarios de una receta. Primero comprueba si el usuario se encuentra con sesión iniciada y, tras esto, permite guardar el comentario.

Algo muy similar ocurre para las respuestas a los comentarios:

```

class RespuestaController extends Controller {

    public function store(Request $request) {
        $request->validate([
            'id_comentario' => 'required|exists:comentarios,id',
            'contenido' => 'required|string|max:1000',
            'id_receta' => 'required|exists:recetas,id',
        ]);

        Respuesta::create([
            'id_comentario' => $request->id_comentario,
            'id_user' => Auth::id(),
            'contenido' => $request->contenido,
            'id_receta' => $request->id_receta,
            'id_user_respondido' => $request->id_user_respondido,
            'f_creacion' => now(),
        ]);

        return back()->with('success', 'Respuesta publicada correctamente.');
```

Ilustración 57 Función para responder a un comentario

Crear receta

Para crear una nueva receta, se creó un modal que se superpone a la vista en la cual se encuentre el usuario. La idea es facilitar la creación de nuevas recetas de forma fácil, por ello se permite realizar desde cualquier parte de la web, pues no redirige a ninguna vista si no que a través de un modal se crea una receta de forma rápida y sencilla.

Ilustración 58 Formulario de creación/edición de receta

El modal aparece a través de un script al pulsar el botón de “Crear receta” que aparece siempre en la barra de navegación. El guardado de la receta en la base de datos se maneja desde el controlador de recetas a través del siguiente método, que realiza todas las comprobaciones pertinentes (comprueba si ya existe la misma receta con la misma ID, si el usuario está autorizado a crearla, etc.) para realizar la acción de guardado:

```
public function guardarRecetaUsuarioAjax($id)
{
    $userId = Auth::id();

    // Verificar si ya está guardada
    $yaExiste = GuardarReceta::where('id_receta', $id)->where('id_user', $userId)->exists();

    if (!$yaExiste) {
        GuardarReceta::create([
            'id_receta' => $id,
            'id_user' => $userId,
            'f_guardar' => now(),
        ]);

        return response()->json(['status' => 'success', 'message' => 'Guardada']);
    }

    return response()->json(['status' => 'failed', 'message' => 'Ha ocurrido un error']);
}
```

Ilustración 59 Función de guardado de receta

Perfil de usuario

En esta vista se muestra toda la información del perfil del usuario. Para acceder a la información del propio usuario que ha iniciado sesión, simplemente se puede acceder pulsando el botón de “Perfil” de la barra de navegación. Por otro lado, también se puede acceder a perfiles de otros usuarios, para visualizarlos, ver las recetas que han creado y las que les gustan e incluso su lista de seguidores y personas que siguen. Se puede editar el perfil del usuario propio, pero no el de los demás y esto tiene un estricto control de seguridad para evitarlo.

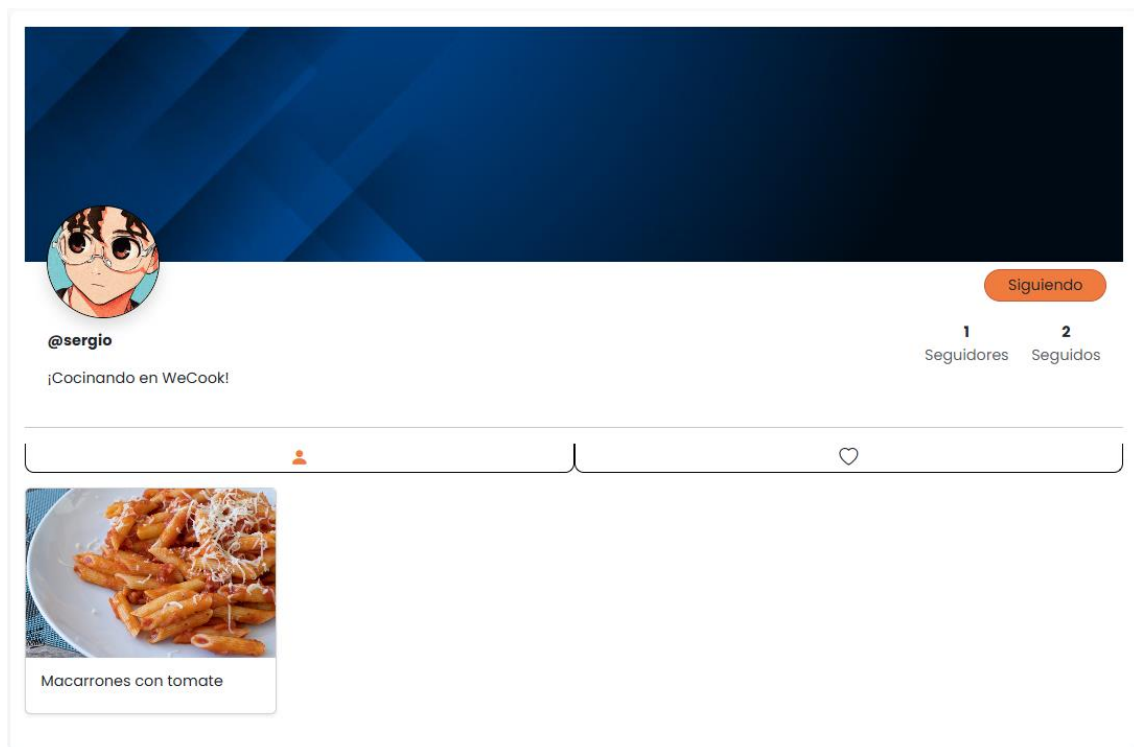


Ilustración 60 Perfil de usuario

Todas las funcionalidades relacionadas con el perfil se manejan desde su propio controlador. En concreto, el visualizar la vista de perfil se maneja desde el siguiente método:


```

class ProfileController extends Controller {

  public function ver($id)
  {
    $perfil = Perfil::where('id_user', $id)->firstOrFail();

    $seguido = false;

    if (Auth::check()) {
      $userId = Auth::id();
      $seguido = SeguirUsuario::where('id_user', $id)
        ->where('id_seguidor', $userId)
        ->exists();
    }

    // Enviar todo a la vista
    return view('profile.perfil', compact('perfil', 'seguido'));
  }
}

```

Ilustración 61 Función para ver perfil de usuario

Los botones para ver la lista de seguidores y seguidos se controlan desde estos métodos:

```

public function verSeguidores($id)
{
  $perfil = Perfil::where('id_user', $id)->firstOrFail();
  $usuario = $perfil->user;

  $seguidores = $usuario->seguidores()->with('perfil')->get();

  return view('profile.seguidores', compact('perfil', 'seguidores'));
}

public function verSeguidos($id)
{
  $perfil = Perfil::where('id_user', $id)->firstOrFail();
  $usuario = $perfil->user;

  $seguidos = $usuario->seguidos()->with('perfil')->get();

  return view('profile.seguidos', compact('perfil', 'seguidos'));
}

```

Ilustración 62 Función para ver seguidores y seguidos de usuario

Y estas son las vistas que muestran (ambas son iguales, por lo que se muestra una de ejemplo):

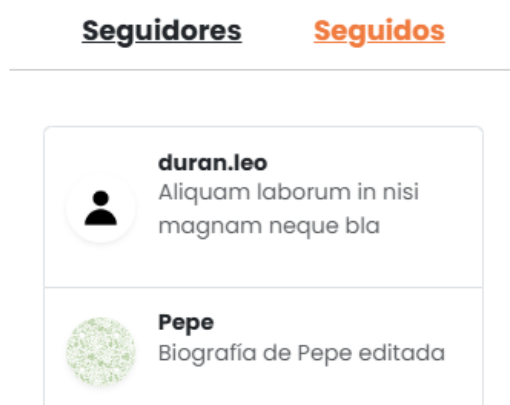


Ilustración 63 Seguidores/seguidos de usuario

La edición del perfil se maneja desde este método:

```
public function actualizar(Request $request, $id)
{
    $request->validate([
        'nombre' => 'required|string|max:255',
        'descripcion' => 'nullable|string',
        'img_perfil' => 'nullable|image|mimes:jpeg,png,jpg|max:2048',
        'img_banner' => 'nullable|image|mimes:jpeg,png,jpg|max:4096',
    ]);

    $perfil = Perfil::where('id_user', $id)->firstOrFail();

    $perfil->name = $request->nombre;
    $perfil->biografia = $request->descripcion;

    if ($request->hasFile('img_perfil')) {
        $perfil->img_perfil = $request->file('img_perfil')->store('perfiles', 'public');
    }

    if ($request->hasFile('img_banner')) {
        $perfil->img_banner = $request->file('img_banner')->store('perfiles', 'public');
    }

    $perfil->save();

    return redirect()->route('perfil.ver', ['id' => $perfil->id_user])
        ->with('success', 'Perfil actualizado correctamente.');
```

Ilustración 64 Función de actualizar perfil de usuario

Y esta es el modal donde se editan los datos del perfil:

Ilustración 65 Formulario de edición de perfil de usuario

Por un lado, el botón de editar perfil muestra un modal que permite cambiar la información e imágenes del perfil con facilidad, y el método anterior es el que controla la actualización de dichos datos en la base de datos con sus correspondientes validaciones de seguridad.

En cuanto a la visualización de las tarjetas de las recetas creadas y gustadas por un usuario, se realizan a través de AJAX, pues de esta forma se permite cambiar entre ambos listados en el perfil al pulsar los diferentes botones sin tener que recargar la interfaz completa.

Recetas guardadas

Esta es la vista donde se muestran las recetas guardadas por un usuario. Estas no son las recetas creadas por un usuario ni las que marcó como gustadas, si no una vista adicional que muestra aquellas que fueron marcadas como guardadas. Es una funcionalidad pensada para guardar aquellas recetas que no se quieren dar constancia de que se guardaron para el resto de los usuarios como sí ocurre con las recetas gustadas. Se trata de un apartado más privado.

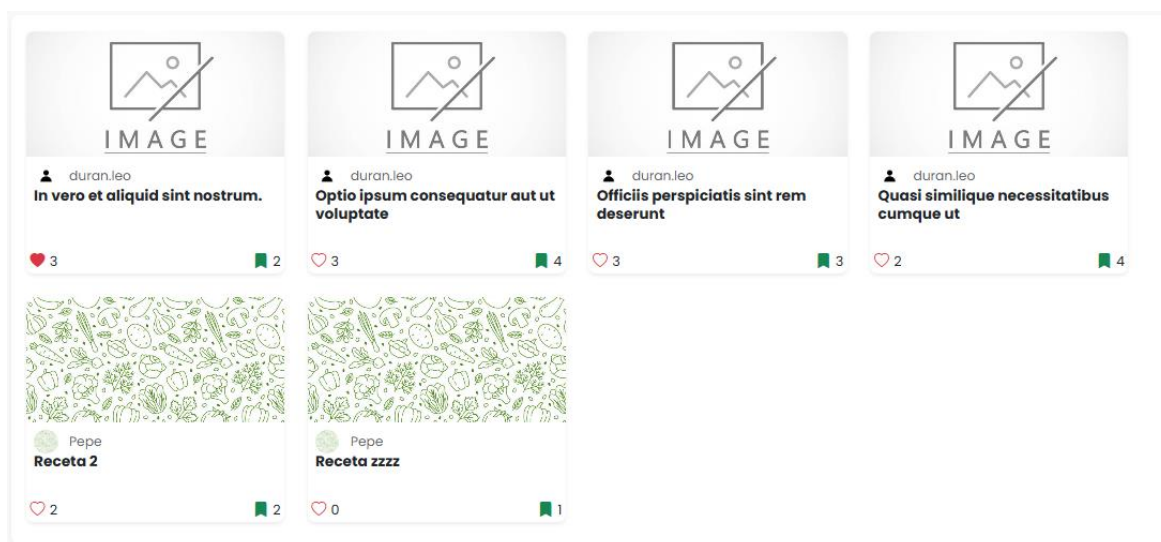


Ilustración 66 Lista de recetas guardadas

Todo esto se controla a través de este método del controlador de recetas:

```
public function listarRecetasGuardadasAjax(Request $request){  
  
    $guardadas = GuardarReceta::where('id_user',Auth::id())->select('id_receta')->get();  
    $recetas = Receta::whereIn('id', $guardadas)->get();  
  
    foreach($recetas as $r){  
  
        $r['meGustas'] = count($r->usuariosQueGustaron);  
        $r['vecesGuardados'] = count($r->usuariosQueGuardaron);  
        $r['nombreAutor'] = $r->autor->perfil->name;  
  
        if(GustarReceta::where('id_receta',$r->id)->where('id_user',Auth::id())->exists()){  
            $r['like'] = true;  
        }  
        else  
        {  
            $r['like'] = false;  
        }  
    }  
  
    return response(json_encode($recetas),200)->header('Content-type','text/plain');  
}
```

Ilustración 67 Función para listar recetas guardadas

Muestra el listado con las recetas guardadas del usuario, permitiendo incluso marcar o desmarcar como gustadas o guardadas, actualizándose dinámicamente mediante AJAX.

Ajustes de cuenta

En esta vista se le da la posibilidad de cambiar su dirección de correo electrónico o su contraseña al usuario:

⚙ Ajustes de usuario

✉ Cambiar email

@ Nuevo email

✓ Actualizar email

🔒 Cambiar contraseña

🛡 Contraseña actual

🔑 Nueva contraseña

➡ Confirmar nueva contraseña

✓ Actualizar contraseña

Ilustración 68 Formulario de ajustes de usuario

Esta parte del código es la que se encarga de redirigir a dicha vista a través de su ruta cuando se pulsa el botón correspondiente:

```
public function ajustesCuenta()
{
    return view('profile.cuenta');
}
```

Ilustración 69 Ruta al formulario de ajustes de usuario

Y desde este apartado del controlador se gestionan los cambios en el usuario:

```

public function actualizarEmail(Request $request)
{
    /** @var \App\Models\User $user */
    $user = auth()->user(); Undefined method 'user'.

    $request->validate([
        'email' => 'required|email|unique:users,email,' . $user->id,
    ]);

    $user->update([
        'email' => $request->email,
        'email_verified_at' => null,
    ]);

    return back()->with('success', 'Email actualizado correctamente. Por favor, verifica tu nuevo email.');
```

```

public function actualizarPassword(Request $request)
{
    $user = auth()->user(); Undefined method 'user'.

    $request->validate([
        'current_password' => ['required', 'current_password'],
        'password' => ['required', 'confirmed', Rules\Password::defaults()],
    ]);

    $user->update([
        'password' => Hash::make($request->password),
    ]);

    return back()->with('success', 'Contraseña actualizada correctamente.');
```

Ilustración 70 Funciones para actualizar en ajustes de usuario

Por un lado, tenemos la función que maneja el cambio de correo. Primero, verifica que el usuario ha iniciado sesión y en dicho caso, comprueba que el nuevo correo es válido y no se repite en la base de datos. A continuación, realiza la actualización y muestra un mensaje al usuario dependiendo si la acción se ha realizado correctamente o no.

Por otro lado, la función para cambiar la contraseña realiza sus validaciones pertinentes. Comprueba que el usuario ha iniciado sesión y en dicho caso, le permite escribir una nueva contraseña con la que iniciar sesión y muestra un mensaje dependiendo si la acción se ha realizado con éxito o no. Dicha contraseña, al igual que en el registro de usuario, recibe una codificación hash para mantener la seguridad en la base de datos.

Pie de página

El pie de página se sitúa en la parte inferior derecha, en forma de botón desplegable para mantener una estética limpia y simple. En este, se encuentran enlaces a diferentes vistas relacionadas con información adicional sobre la aplicación y otras funcionalidades similares.

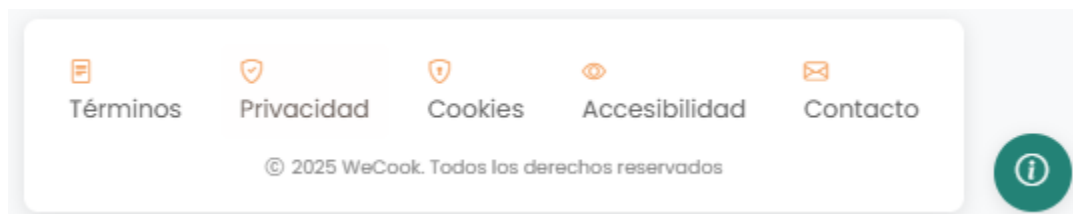


Ilustración 71 Pie de página de la web

Todas las rutas de estas vistas se gestionan a través del gestor de rutas de la aplicación, puesto que tiene sentido crear un controlador simplemente para redirigir a vistas que contienen texto informativo:

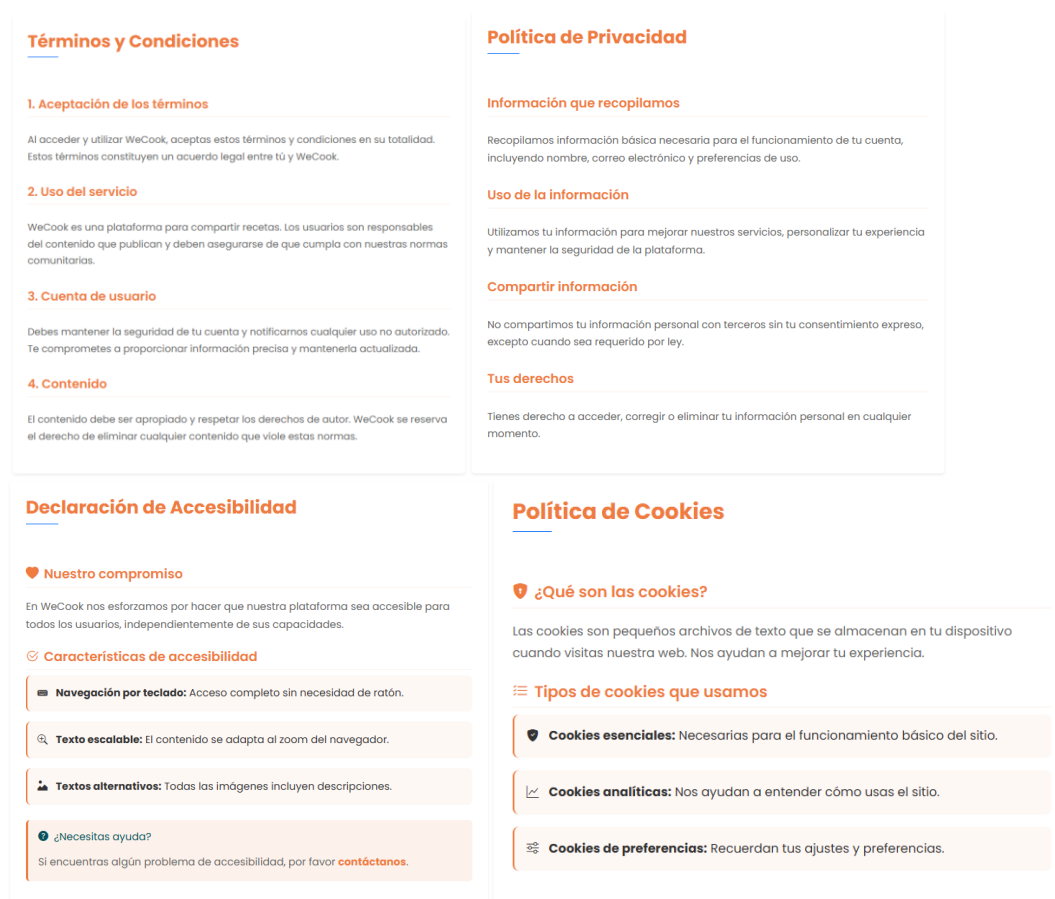
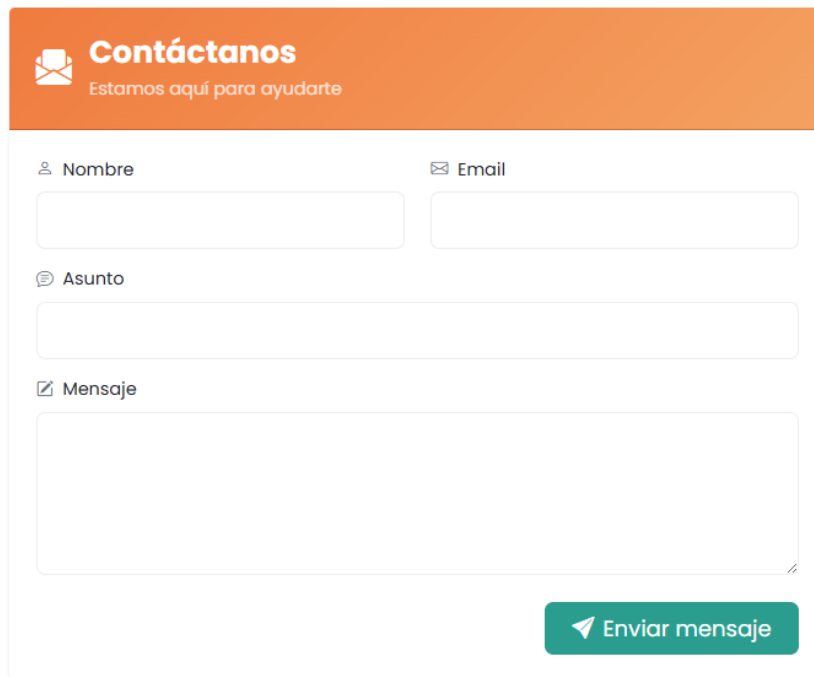


Ilustración 72 Vistas de información legal de la plataforma

La única excepción de estas vistas es la de contacto, puesto que tiene la funcionalidad de enviar un correo electrónico a la dirección de recepción de correos de la aplicación.



Contáctanos
Estamos aquí para ayudarte

Nombre

Email

Asunto

Mensaje

Enviar mensaje

Ilustración 73 Formulaio de contacto con soporte técnico

La funcionalidad de esta vista se gestiona desde un controlador propio específico para ella misma:

```
class MailController extends Controller
{
    public function index()
    {
        return view('legal.contacto');
    }

    public function enviar(Request $request)
    {
        $request->validate([
            'nombre' => 'required|string|max:255',
            'email' => 'required|email|max:255',
            'asunto' => 'required|string|max:255',
            'mensaje' => 'required|string',
        ]);

        $datos = $request->only('nombre', 'email', 'asunto', 'mensaje');

        // Enviar correo
        Mail::to('contacto@wecook.com')->send(new ContactoMail($datos));

        return redirect()->route('contacto')->with('success', 'Mensaje enviado correctamente.');
```

Ilustración 74 Función que maneja envío de correo de contacto

Realiza validación de los datos enviados y comprueba que cumpla con el tipo de dato requeridos y, en caso de ser correcto, muestra un mensaje y envía dichos datos a la dirección de correo predeterminada de la aplicación para que el equipo de desarrollo pueda leerlo y tratar de ayudar al usuario.

Administración de recetas y usuarios

La vista de administrador permite gestionar a los usuarios de la plataforma y a las recetas guardadas en la misma. Principalmente, permite el control de todo esto, pues puede ser necesario moderar contenido indeseable o que incumpla las normas de la aplicación, así como poder resolver problemas concretos.

Por un lado, encontraremos el listado de recetas, mostradas dinámicamente a través de AJAX:

Recetas

Recetas

Usuarios

Mostrar 10 Entradas

Buscar:

ID	Título	Tipo	Autor	Estado	Fecha Creación	
1	Perspiciatis facere cum autem quasi.	Cena	durán.leo	Pública	14-06-2025	Ocultar Eliminar
2	In vero et aliquid sint nostrum.	Desayuno	durán.leo	Pública	14-06-2025	Ocultar Eliminar
3	Optio ipsum consequatur aut ut voluptatem praesentium suscipit.	Desayuno	durán.leo	Pública	14-06-2025	Ocultar Eliminar
4	Exercitationem ea voluptatem itaque voluptatem hic.	Almuerzo	durán.leo	Pública	14-06-2025	Ocultar Eliminar
5	Officiis perspiciatis sint rem deserunt dolore ut delectus quaerat.	Almuerzo	durán.leo	Pública	14-06-2025	Ocultar Eliminar
6	Ratione ut facilis iure eum adipisci facere.	Postre	durán.leo	Pública	14-06-2025	Ocultar Eliminar
7	Omnis aut voluptate laborum commodi voluptatem at.	Cena	durán.leo	Pública	14-06-2025	Ocultar Eliminar
8	Beatae et eos expedita dignissimos.	Postre	durán.leo	Pública	14-06-2025	Ocultar Eliminar
9	Deserunt distinctio voluptatum numquam velit.	Cena	durán.leo	Pública	14-06-2025	Ocultar Eliminar
10	Quasi similique necessitatibus cumque ut earum rerum dolor.	Almuerzo	durán.leo	Pública	14-06-2025	Ocultar Eliminar

Mostrando 1 a 10 de 25 Entradas

Primero Anterior 1 2 3 Siguiente Ultimo

Ilustración 75 Administración de recetas

Por otro, encontramos un listado similar de usuarios también cargado dinámicamente:

Recetas

Usuarios

Usuarios

Mostrar

10

Entradas

Buscar:

ID	Email	Rol	Recetas Creadas	Fecha	Acciones
1	almanza.pau@example.net	Usuario	0	28-05-2025	<div>Ascender</div> <div>Eliminar</div>
2	cvicente@example.net	Usuario	0	28-05-2025	<div>Ascender</div> <div>Eliminar</div>
3	gerard73@example.com	Admin	20	28-05-2025	<div>Degradar</div> <div>Eliminar</div>
4	gerard.pagan@example.com	Usuario	0	28-05-2025	<div>Ascender</div> <div>Eliminar</div>
5	tgaltvan@example.com	Usuario	0	28-05-2025	<div>Ascender</div> <div>Eliminar</div>
6	andres.marquez@example.org	Usuario	0	28-05-2025	<div>Ascender</div> <div>Eliminar</div>
7	jimenez.yaiza@example.com	Usuario	0	28-05-2025	<div>Ascender</div> <div>Eliminar</div>
8	valentina48@example.net	Usuario	0	28-05-2025	<div>Ascender</div> <div>Eliminar</div>
9	noa.gracia@example.com	Usuario	0	28-05-2025	<div>Ascender</div> <div>Eliminar</div>
10	briones.juana@example.net	Usuario	0	28-05-2025	<div>Ascender</div> <div>Eliminar</div>

Mostrando 1 a 10 de 15 Entradas

Primero

Anterior

1

2

Siguiente

Ultimo

Ilustración 76 Administración de usuarios

Esto se maneja a través del controlador de administrador. Este fragmento del código redirige a la vista:

```
public function index(Request $request)
{
    return view('admin.admin');
}
```

Ilustración 77 Ruta a vista de administración

Este otro fragmento se encarga de recoger el listado de recetas para mostrarlas mediante AJAX:

```
public function listaRecetasAjax(Request $request)
{
    $recetas = Receta::query();

    return Datatables::eloquent($recetas) // Le mando la query al Datatable

    // Hago que la columna de fecha se muestre como yo quiero
    ->addColumn('created_at', function($receta){
        return Carbon::parse($receta->created_at)->format('d-m-Y');
    })

    // Hago que en vez del id me muestre el autor
    ->addColumn('autor_receta', function($receta){
        return $receta->autor->perfil->name;
    })

    // Añado una columna de acciones y creo los botones que quiera
    ->addColumn('action', function($receta){
        return '<button data-id="'.$receta->id.'" class="btn btn-danger btn-sm delete-receta">Eliminar</button>';
    })

    ->rawColumns(['action'])

    ->make(true);
}
```

Ilustración 78 Función que lista tabla de recetas para administración

Y este otro lo mismo, para el listado de usuarios:

```
public function listaUsuariosAjax(Request $request)
{
    $usuarios = User::query();

    return Datatables::eloquent($usuarios)

    ->addColumn('created_at', function($user){
        return Carbon::parse($user->created_at)->format('d-m-Y');
    })

    ->addColumn('action', function($user){
        return '<button data-id="'.$user->id.'" class="btn btn-danger btn-sm delete-user">Eliminar</button>';
    })

    ->rawColumns(['action'])

    ->make(true);
}
```

Ilustración 79 Función que lista tabla de usuarios para administración

Ambas funciones funcionan de manera similar; obtienen el listado de la base de datos a través del modelo y añade botones de funcionalidad adicionales para cada fila (como el botón de eliminar) para permitir la administración de dichos datos.

Además, cada tabla añade también una paginación que facilita la navegación e incluso un filtrado dinámico para cada columna.

7. EVOLUCIÓN DEL PROYECTO Y TRABAJO FUTURO

Evolución del proyecto

A lo largo del proyecto se han producido numerosos cambios en referencia al concepto inicial que se concibió. Dichos cambios se produjeron, principalmente, a nivel de diseño. Al estar en el proceso de desarrollo, nos dimos cuenta de que a nivel de diseño podíamos mejorar ciertos elementos (como la ubicación de la barra de navegación y los colores principales de la aplicación) que se adaptaban mejor al tipo de aplicación que estábamos creando.

Surgieron también otras mejoras a nivel funcional, como ligeros cambios en la base de datos para ajustarse a las necesidades que se nos presentaron (relaciones entre ciertas tablas que no se tuvieron en cuenta en un inicio, etc.).

Además, al ir realizando estos cambios, surgieron nuevas ideas y posibilidades, por lo que pudimos implementar nuevas funcionalidades al proyecto, como nuevas herramientas para los perfiles de administración de la aplicación o el envío de correos electrónicos (lo cual no se concibió al inicio del proyecto).

Trabajo futuro

Tras todo el trabajo realizado y con el estado actual de la aplicación consideramos que, a pesar de que tenga una funcionalidad completa y correcta, hay ideas de mejora y de ampliación que consideramos que elevarían la calidad del proyecto:

- Mejoras en las alertas y mensajes: mejorar la forma en la que la aplicación muestra mensajes o advertencias al usuario cuando realiza alguna acción. Aunque se muestran multitud de ellas, consideramos que este apartado es mejorable, pues es muy importante la información que recibe el usuario al interactuar con la aplicación.
- Sistema de bloqueo y silencio de usuarios: actualmente, solo los administradores pueden bloquear a otros usuarios y de forma global en la aplicación. Una funcionalidad muy frecuente en las redes sociales es la de permitir a cada usuario bloquear o silenciar a otros usuarios de manera personal. De esta forma, se crea un entorno más seguro y agradable para los usuarios.
- Mejoras en la base de datos: actualmente, al eliminar información, se elimina por completo de la base de datos. Sería conveniente mejorar la estructura de la base de datos para poder crear nuevas tablas y relaciones que guarden un registro de aquella información que se quiere borrar de la aplicación (por ejemplo, eliminar una receta) y que en lugar de eliminarla

de manera completa, se guarde en otra tabla con aquella información que se pretendía eliminar. Así, se mejorarían los registros por si se generasen conflictos de interés o incluso de ámbito legal.

- **Diseño:** aunque el diseño mejoró considerablemente respecto al planteado al inicio, aún hay margen de mejora para conseguir un diseño más limpio y elegante, que sea más llamativo y cómodo a nivel visual para los usuarios.
- **Código:** a nivel de programación hay también margen de mejora, pues se pueden optimizar algunos fragmentos algo repetitivos con estructuras de bucles y una escritura más clara y sencilla de algunas partes del código que faciliten el mantenimiento a futuro de la aplicación.
- **Despliegue:** una de las intenciones era desplegar la aplicación para que quien quisiera pudiera encontrarla navegando por internet y darle un uso real. Sin embargo, por falta de tiempo y por poner el foco en otros elementos que considerábamos más importantes para la funcionalidad de la aplicación, no hemos podido hacer esta parte a tiempo.

8. CONCLUSIONES

La realización de este proyecto ha supuesto un gran reto y una excelente oportunidad de aprendizaje. Laravel, aunque completamente nuevo para nosotros al comienzo del desarrollo, ha demostrado ser una herramienta muy útil, bien documentada y con una comunidad activa que facilita el aprendizaje.

Gracias a su estructura basada en el patrón MVC (el cual ya conocíamos) y sus numerosas funcionalidades integradas, hemos podido comprender mejor cómo se construye una aplicación web profesional desde cero, siendo además un trabajo más cómodo de realizar que con PHP simple.

También hemos mejorado nuestros conocimientos de bases de datos relacionales, seguridad en aplicaciones web y diseño de interfaces. Además, durante el proceso de desarrollo tuvimos que cambiar la idea y el diseño de algunos elementos, pues surgieron mejores ideas y conceptos nuevos que al inicio de este no conocíamos.

En definitiva, este proyecto nos ha permitido aplicar conocimientos teóricos adquiridos a lo largo del grado y desarrollar competencias clave para nuestro futuro profesional en el ámbito del desarrollo web.

9. AGRADECIMIENTOS

A Hugo María Vegas Carrasco en primer lugar, por tutorizar este proyecto, su cercanía y su ayuda durante todo el desarrollo del proyecto y a Ana Sosa Serrano por idear y diseñar el logotipo de esta aplicación y del proyecto.

10. BIBLIOGRAFÍA

No hay bibliografía consultada.

11. WEBGRAFÍA

Bootstrap. (s. f.). <https://getbootstrap.com/>

Laravel. (s. f.). <https://laravel.com/docs/12.x>

MySQL Documentation. (s. f.). <https://dev.mysql.com/doc/>

Stack Overflow. (s. f.). <https://stackoverflow.com/>

W3Schools. (s. f.). <https://www.w3schools.com/>

Composer. (s. f.). <https://getcomposer.org/doc/>

Laravel Breeze. (s. f.). <https://laravel.com/docs/12.x/starter-kits#breeze>

Font Awesome. (s. f.). <https://fontawesome.com/docs>

Google Fonts. (s. f.). <https://fonts.google.com/>

12. ANEXOS

Manual técnico

REQUISITOS

Para poder instalar y arrancar la aplicación, es necesario tener instalado en el sistema:

- PHP 8.2 o superior. Sirven tanto PHP en solitario como PHP que viene incluido en XAMPP, siempre que sea la versión 8.2 o superior. Puedes descargar cualquiera de los 2 desde las páginas oficiales:

XAMPP: <https://www.apachefriends.org/download.html>

PHP: <https://www.php.net/downloads.php>

Puedes comprobar que tienes PHP instalado correctamente con el siguiente comando:

php -v

```
C:\Users\Sergio>php -v
PHP 8.4.3 (cli) (built: Jan 15 2025 11:03:08) (NTS Visual C++ 2022 x86)
Copyright (c) The PHP Group
Zend Engine v4.4.3, Copyright (c) Zend Technologies
```

- MySQL. Puedes descargar MySQL Workbench (cliente de base de datos con MySQL) en este enlace: <https://dev.mysql.com/downloads/windows/installer/8.0.html>

Nota: no es necesario instalar Workbench, pero es recomendable para gestionar la base de datos ya que facilita el trabajo.

- Git. Necesario para clonar y actualizar el repositorio. Descargar del siguiente enlace: <https://git-scm.com/downloads/win>
- Composer. Instalar Composer desde este enlace (descargar y ejecutar el .exe): <https://getcomposer.org/download/>

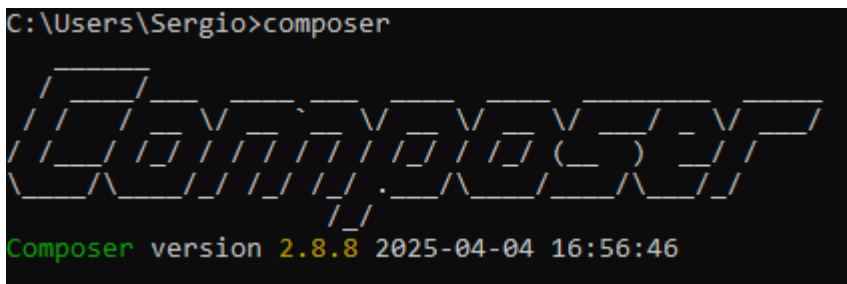
Al instalar, aceptar y continuar con los ajustes predeterminados.

Para comprobar que se ha instalado correctamente:

Composer --v

Nota: es posible que al tratar de instalar Composer surja algún error.

Para instalarlo, es recomendable desactivar el antivirus y así evitar cualquier problema (es completamente seguro).

A screenshot of a terminal window with a black background. The prompt is 'C:\Users\Sergio>composer'. Below the prompt is a large, stylized logo for 'Composer' made of white and yellow lines. At the bottom, it says 'Composer version 2.8.8 2025-04-04 16:56:46' in green and yellow text.

```
C:\Users\Sergio>composer

Composer version 2.8.8 2025-04-04 16:56:46
```

- Node.js. Ve al sitio oficial: <https://nodejs.org>.

Descarga la versión recomendada (LTS).

Ejecuta el instalador .msi y acepta los pasos por defecto.

Para verificar que se instaló correctamente:

node -v

npm -v

INSTALACIÓN DEL PROYECTO

1. Clonar repositorio

Desde la carpeta donde se quiera ubicar el proyecto, ejecutar en la terminal:

```
git clone https://github.com/sergioderon1803/TFG_RRSS_Cocina.git
```

2. Comprobar que el repositorio está correctamente vinculado

```
git remote -v
```

Debería aparecer algo como esto:

```
origin https://github.com/usuario/repositorio.git (fetch)
```

```
origin https://github.com/usuario/repositorio.git (push)
```

3. Configurar el fichero .env que se encuentra en la raíz del proyecto.

Añadir o modificar esta parte para configurarlo conforme a nuestra base de datos (crear schema en la base de datos con nombre “wecook” si no existe):

```
DB_CONNECTION=mysql
```

```
DB_HOST=127.0.0.1
```

```
DB_PORT=3306
```

```
DB_DATABASE=wecook
```

```
DB_USERNAME=root
```

```
DB_PASSWORD=password
```

4. Dentro de la carpeta del proyecto (/proyecto_recetas), instalar dependencias con composer:

```
composer install
```

```
C:\Docencia\Curso_24_25\Proyectos\RRSS Cocina\TFG_RRSS_Cocina\proyecto_recetas>composer install
Installing dependencies from lock file (including require-dev)
Verifying lock file contents can be installed on current platform.
Nothing to install, update or remove
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

 INFO  Discovering packages.

laravel/breeze ..... DONE
laravel/pail ..... DONE
laravel/sail ..... DONE
laravel/tinker ..... DONE
nesbot/carbon ..... DONE
nunomaduro/collision ..... DONE
nunomaduro/termwind ..... DONE

80 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
```

5. Migrar tablas de la BBDD

php artisan migrate

Para insertar datos de prueba en la base de datos (opcional):

php artisan db:seed

Nota: para migrar la base de datos con datos en un solo comando ejecutar

php artisan migrate:fresh --seed

6. Arrancar aplicación

php artisan serve

7. Instalar dependencias con npm:

npm install

8. Compilar los assests (js, css, etc.)

npm run build

10. Acceder a la aplicación a través de <http://localhost:8000/>

EXTRA. Instalación de otras funcionalidades complementarias del proyecto

Iconos de Bootstrap: `npm install bootstrap-icons`

Funcionamiento por AJAX: `composer require yajra/laravel-datatables-oracle`

Almacenamiento y visualización de imágenes: `php artisan storage:link`

PARA DESARROLLADORES

- Recuperar actualizaciones del repositorio:

```
git pull
```

- Actualizar cambios del repositorio local:

```
git add . o git add "nombre_archivo"
```

- Guardar cambios

```
git commit -m "Mensaje de resumen de los cambios":
```

- Subir cambios desde repositorio local hacia repositorio remoto:

```
git push
```

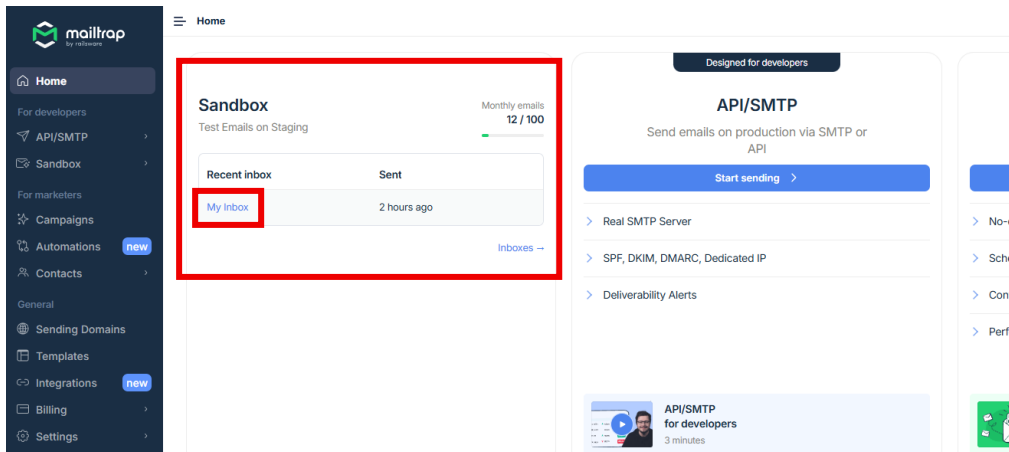
Nota: puesto que estamos trabajando sobre la rama main directamente, asegurarse de tener la última actualización en local antes de subir cualquier cambio.

Nota 2: durante el desarrollo, para no compilar assest continuamente mientras se realizan cambios,

usar `npm run dev` (compila mientras se realizan cambios).

- Activar funcionalidad de envía de mails:

Para poder probar las funcionalidades relacionadas con el envío de mails, debemos crearnos un usuario en Mailtrap (<https://mailtrap.io/>) y acceder al siguiente apartado:



Con las credenciales facilitadas en Mailtrap:

Credentials	
Host	sandbox.smtp.mailtrap.io
Port	25, 465, 587 or 2525
Username	8d3e5760215b32
Password	****f3da
Auth	PLAIN, LOGIN and CRAM-MD5
TLS	Optional (STARTTLS on all ports)

Cambiar en el fichero .env MAIL_USERNAME y MAIL_PASSWORD

MAIL_MAILER=smtp

MAIL_SCHEME=null

MAIL_HOST=sandbox.smtp.mailtrap.io

MAIL_PORT=2525

MAIL_USERNAME= (sustituir por username otorgado por Mailtrap)

MAIL_PASSWORD= (sustituir por password otorgado por Mailtrap)

MAIL_ENCRYPTION=null

MAIL_FROM_ADDRESS="no-reply@wecook.test"

MAIL_FROM_NAME="\${APP_NAME}"

Con esto configurado, cada mail que se deba enviar desde la aplicación nos llegará a la bandeja de recibidos de Mailtrap:

