



Diseño e implementación de una plataforma web de recetas con accesibilidad cognitiva

Yummies

Autoras

María del Pilar Navío Colón

Paula Casado García

Grado Superior en Desarrollo de Aplicaciones Web

Curso académico 2020-2022

edix

Introducción, justificación y objetivos

Actualmente, las aplicaciones de cocina están en auge debido a que facilitan la planificación de las dietas diarias en una sociedad con un ritmo de vida acelerado. Sin embargo, gran parte de las recetas propuestas por estas aplicaciones son difíciles de realizar, tanto por el tiempo requerido para realizarlas como por el tipo de ingredientes utilizados.

Por otro lado, cocinar supone una tarea que consume recursos cognitivos, principalmente los relacionados con funciones ejecutivas como son la atención, la memoria de trabajo o la planificación. Esto conlleva una dificultad extra para aquellas personas que padecen algún tipo de discapacidad cognitiva, como la derivada del daño cerebral adquirido o Síndrome de Down. Si se unen las dos problemáticas mencionadas, se puede concluir que la cocina puede convertirse en un gran obstáculo para esta población.

El presente proyecto tiene como objetivo salvar ese obstáculo. Consiste en una aplicación web, llamada *Yummies*, que ofrece una variedad de recetas usando el ‘stock’ personal de ingredientes que hay en la nevera para lograr una cocina sencilla y sostenible, mediante un diseño basado en las pautas de accesibilidad cognitiva.

Objetivos

El objetivo principal de la web es conseguir inclusividad, es decir, que cualquier persona pueda optimizar los recursos alimenticios que estén a su alcance. Por lo tanto, un aspecto a destacar de la aplicación es la adaptación a las pautas de **accesibilidad cognitiva**, siguiendo los **estándares de W3C de usabilidad para personas con discapacidad cognitiva**: alteración de las funciones ejecutivas (atención, memoria de trabajo, etc), discapacidad intelectual, trastorno del espectro autista, deterioro cognitivo o demencia, trastorno con déficit de atención e hiperactividad (TDAH), trastornos del lenguaje, daño cerebral adquirido...

Entre las pautas a seguir para conseguir accesibilidad cognitiva destacan:

- El nivel de destreza culinaria no será un factor determinante, puesto que habrá recetas con diferente grado de dificultad
- Ayudar a los usuarios a entender qué es cada cosa y cómo usarla

- Ayudar a los usuarios a encontrar lo que necesitan
- Utilizar un contenido claro y comprensible
- Ayudar a los usuarios a evitar errores o a corregirlos
- Ayudar a los usuarios a mantener la atención
- Los procesos no dependen de la memoria, principalmente de la memoria de trabajo (por ejemplo, aportar feedback continuo a medida que se rellena un formulario de forma que el usuario no necesite recordar las instrucciones)

Objetivos específicos

- Objetivos de la aplicación:
 - Implementar una aplicación web para la consulta y publicación de recetas.
 - Facilitar la búsqueda de recetas mediante sus ingredientes o nombre de la receta.
 - Proporcionar la cantidad de ingredientes en función del número de comensales indicados.
- Objetivos para el usuario:
 - Búsqueda de recetas adaptadas a sus necesidades cognitivas
 - Crear su propio recetario.
 - Fomentar la autonomía de personas con cierto grado de dependencia.

Palabras clave

- Accesibilidad cognitiva
- API Rest
- Spring Boot
- Spring Data JPA
- React JS
- Recetas de cocina

Índice general

Introducción, justificación y objetivos	2
Objetivos	2
Objetivos específicos	3
Palabras clave	4
Índice general	5
Resumen	6
Módulos formativos aplicados en el trabajo	7
Tecnologías utilizadas	9
Componentes del equipo y aportación realizada por cada estudiante	10
Fases del proyecto	11
Fase de contacto inicial	11
Fase de planificación	11
Fase de diseño gráfico y UI/UX	12
Usabilidad y UX	12
Estilo	18
Fase de desarrollo	21
Modelo de datos utilizado	21
Diseño de la base de datos: diagrama de clases	22
Back-end	23
Fase de pruebas	38
Fase de lanzamiento	40
Conclusiones y mejoras del proyecto	42
Bibliografía	45
Anexos	48
Contenido del proyecto	48
Repositorio de Github	48
Guía de usuario	48

Resumen

El proyecto trata de la creación de una aplicación web para visualizar y dar de alta recetas propias. Es una web accesible para el público con dificultades cognitivas.

A priori se contemplan dos tipos de usuarios: los administradores de la aplicación y usuarios que consumen la aplicación. Estos últimos pueden realizar diferentes acciones:

- Consultar recetas categorizadas.
- Buscar recetas (por nombre de ingrediente o por el nombre de la receta).
- Filtrar los criterios de búsqueda (por nivel de dificultad o por tipo de dieta).
- Visualizar la cantidad de ingredientes necesarios para cada receta en función del número de comensales indicados.
- Dar de alta recetas propias.

Los administradores heredan estos casos de uso, y además, pueden visualizar el listado de los usuarios registrados, ver y dar de alta ingredientes.

El desarrollo de la aplicación ha consistido en la creación de un backend REST API independiente y un frontend utilizando React JS.

Quedan abiertos varios desarrollos a implantar, entre los que destacamos las funcionalidades asociadas al perfil de usuario premium, modificar el perfil del usuario, o editar recetas.

Módulos formativos aplicados en el trabajo

- **Programación I, II:** La aplicación se fundamenta en el paradigma de Programación Orientada a Objetos (POO), y utilizaremos Java como lenguaje de programación.
- **Base de datos I, II:** Uso de bases de datos relacionales.
- **Entornos de desarrollo:** Se ha utilizado un IDE, control de versiones mediante GIT, documentación de la aplicación mediante Javadoc y diagramas UML estructurales y de comportamiento para definir la arquitectura de la aplicación.
- **Lenguaje de Marcas y Sistemas de Gestión de la Información:** Creación de páginas en HTML5 con CSS3.
- **Despliegue de Aplicaciones Web:** Despliegue de la aplicación mediante Tomcat, el cual está incluido en el IDE Eclipse EE.
- **Diseño de Interfaces Web:** Se han aplicado diseños familiares y conocidos para romper las barreras que puedan encontrar personas con diversidad funcional en la navegación por nuestra web. La aplicación es responsive para poder visualizarla desde cualquier dispositivo.
- **Empresa e Iniciativa Emprendedora:** El desarrollo del proyecto no está asociado a ningún Plan de Empresa. No obstante, la idea de desarrollar esta aplicación ha surgido por una necesidad detectada en el mercado, por lo que en otra ocasión podría llevarse a cabo.
- **Desarrollo Web en Entorno Cliente:** Utilizaremos JavaScript a través del framework React JS y los modos de navegar por el DOM para hacer dinámicas las páginas, por ejemplo, para determinar la cantidad de ingredientes en función del número de raciones que se indique.
- **Desarrollo Web en Entorno Servidor:** La aplicación sigue una arquitectura MVC donde las tres capas (una para los datos, otra para la interfaz y la capa lógica) se comuniquen entre ellas. Para el desarrollo del back-end, se ha recurrido a tecnologías utilizadas en esta asignatura.

- **Inglés técnico:** Se han puesto en práctica las destrezas mediante la lectura de información técnica. De momento no se contempla la traducción de la aplicación a otros idiomas.

Tecnologías utilizadas

- Intercambio de información entre el back-end y el front-end mediante una API REST
- Servidor: Apache Tomcat

Front-end

- IDE: Visual Studio Code
- HTML5, CSS3, JavaScript
- Responsive design: Semantic UI React
- Uso de librerías y/o frameworks Javascript: ReactJS, AXIOS

Back-end

- IDE: Eclipse for Java EE developers
- Lenguajes de programación orientada a Objetos: Java EE
- Framework de Java: Spring Boot
- Acceso a base de datos: Spring Data JPA

Base de datos

- Gestor de BD: MySQL Workbench 8.0

Componentes del equipo y aportación realizada por cada estudiante

El equipo de trabajo lo hemos compuesto dos estudiantes del ciclo de grado superior de desarrollo de aplicaciones web.

Ambas integrantes hemos participado en todas las fases del proyecto trabajando de forma paralela en las tareas que requerían mayor esfuerzo. El uso de un sistema de control de versiones, concretamente Git junto con el repositorio remoto de Github, fue fundamental para poder participar simultáneamente en los mismos componentes del proyecto.

Desde el comienzo del proyecto se agendaron sesiones semanales para crear sinergias, hacer revisión de los avances, buscar solución a aquellas situaciones problemáticas, estimar la complejidad de los requisitos, definir las nuevas tareas decidiendo cómo se van a realizar y autoasignárnoslas.

Fases del proyecto

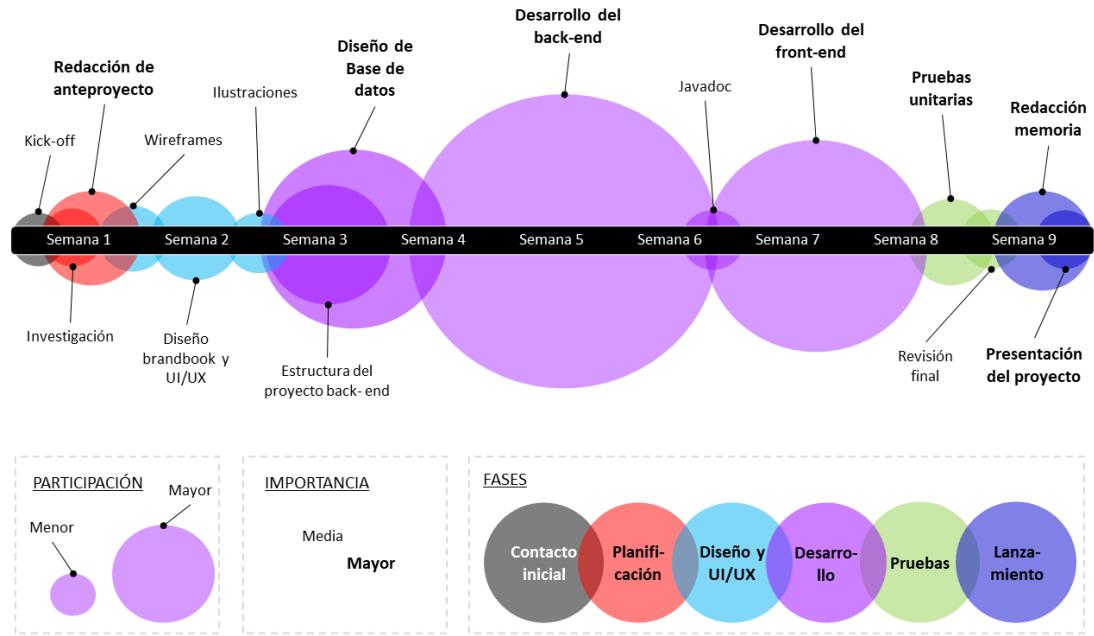


Figura 1: Fases del proyecto

Fase de contacto inicial

En la fase más inicial pusimos en común las ideas para desarrollar el proyecto. La kick-off la concluimos partiendo de la base de que nuestro proyecto tenía que responder a la necesidad de favorecer la inclusión y autonomía de personas con discapacidad cognitiva.

Del mismo modo, acordamos que la temática de la aplicación debía estar relacionada con una necesidad básica y vital para dar cabida a un colectivo numeroso y con un amplio espectro de características. Una aplicación de recetas de cocina cumplía con todos los requisitos.

Fase de planificación

Teniendo definida la temática, nos dispusimos a investigar cuáles eran los medios y tecnologías necesarias y el alcance que conllevaba el desarrollo del proyecto.

La versión beta de la aplicación está disponible para dos tipos de usuarios: administrador y usuario. En el siguiente gráfico se muestran los diferentes casos de uso pensados para cada usuario.

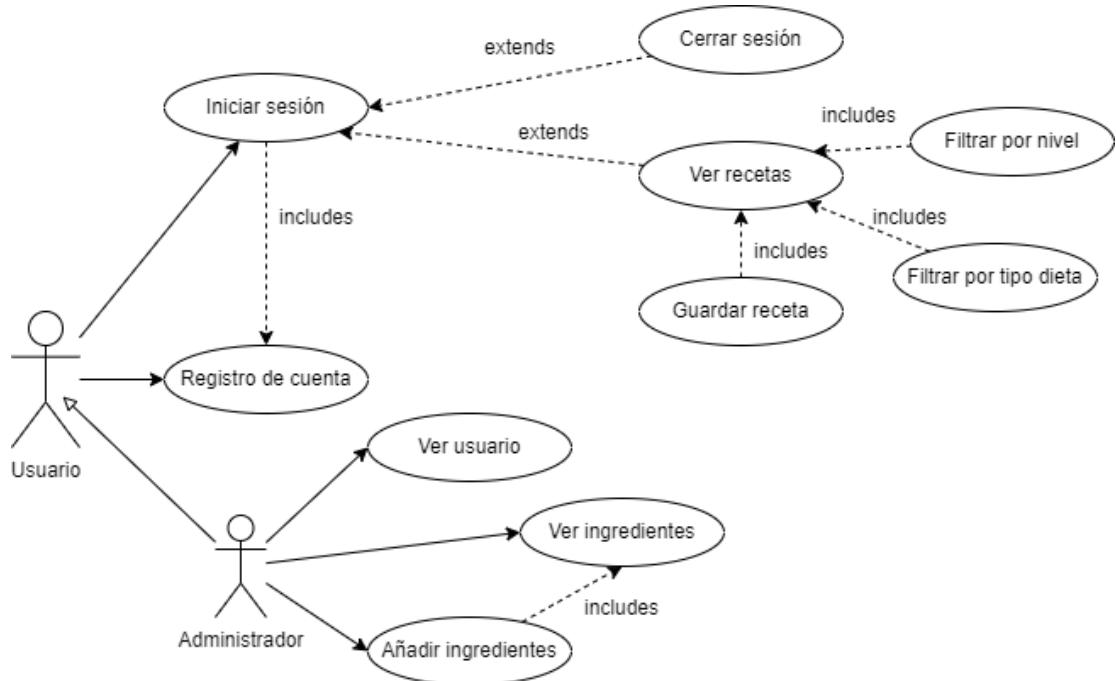


Figura 2: Diagrama UML de los casos de uso

El usuario puede:

- Iniciar sesión, lo que implica previamente haber registrado su cuenta.
 - Una vez que ha iniciado sesión, puede ver las recetas, guardarlas como favoritas y filtrarlas según varios criterios.

El administrador, además de los casos de uso que hereda del perfil usuario puede:

- Ver los usuarios registrados.
 - Ver y dar de alta nuevos ingredientes.

El resultado de esta fase de planificación se recogió en el anteproyecto.

Fase de diseño gráfico y UI/UX

Usabilidad y UX

Las interfaces han guiado el diseño tanto del back-end como del front-end, ya que han permitido identificar los datos que han de recuperarse de la base de datos y los

métodos que muestran dicha información en las diferentes vistas. Para el diseño, se han seguido los **estándares de accesibilidad cognitiva del World Wide Web Consortium (W3C)**.

❖ Distribución y organización de los elementos en las vistas

Una de las premisas que se han llevado a cabo es utilizar un patrón visual coherente, donde se mantenga el mismo estilo en todas las páginas que facilite la navegación por la aplicación. Los patrones de diseño que se han empleado son:

- El flujo de información es principalmente **vertical**
- Las páginas tienen un título
- Un texto descriptivo ayuda a la navegación por la página
- El botón para volver a la página principal se encuentra en la misma ubicación
- El texto de los botones sigue el patrón “*Pulsa aquí para* + acción que realiza el botón en infinitivo” y están acompañados de iconos que ayudan a entender su significado
- Espacio en blanco alrededor de los objetos y textos para separar claramente cada sección y facilitar la lectura para aquellos usuarios con dificultades de comprensión lectora
- Se evitan aquellos elementos que ocultan información, como los desplegables o pestañas
- Se evita el excesivo uso del *Scroll* para reducir la carga cognitiva

❖ Formulario para dar de alta un usuario y una receta

Los estándares de accesibilidad cognitiva del World Wide Web Consortium (W3C) indican la necesidad de ofrecer feedback continuo sobre las acciones del usuario. Es por esta razón que se utiliza el color verde para señalar que el usuario está completando correctamente un input, o se muestra un mensaje informativo en color rojo en caso contrario.

Es necesario que sus campos estén agrupados por temática (datos personales, intereses de cocina, pagos, etc). Cada campo va acompañado de una etiqueta e

instrucciones, pero no contiene placeholders, ya que el usuario puede pensar que ya ha sido rellenado. Las etiquetas de los campos obligatorios, además, van acompañadas de la palabra ***obligatorio***, en lugar de proporcionar la información solamente en las instrucciones iniciales.

Se deben evitar el uso de fechas (ya que los números requieren memoria de trabajo y suponen un desafío para los usuarios con discalculia) e inputs de tipo select, por lo que se deben sustituir por radio buttons o checkboxes.

The wireframe shows a registration form titled "Registro de cuenta". It includes a back arrow icon and a link to return to the login page. There are three input fields: "Escribe tu nombre (requerido)", "Escribe un nombre que no se repita (requerido)", and "Escribe una contraseña (requerido)". A large button at the bottom says "Pusa aquí para crear la cuenta".

Figura 3: Wireframe de la vista de la página de registro de nuevo usuario

❖ Formulario para iniciar sesión (Login)

Es necesario proporcionar un inicio de sesión que no dependa de la memoria u otras capacidades cognitivas, por lo que el usuario no tendrá que memorizar cadenas de caracteres, realizar cálculos, puzzles o reconocer caracteres presentados en la pantalla y escribirlos posteriormente en un input. Además, al tratarse de un formulario, cumple las pautas mencionadas en el apartado anterior.

Existen diferentes propuestas del W3C para conseguir un login accesible, como son la autenticación biométrica (reconocimiento facial o de huellas digitales), usar otro sistema en el que el usuario ya haya iniciado sesión (Google, Facebook, etc.) o la autenticación de doble factor. Para este proyecto, se han implementado los recursos

copiar y pegar la contraseña (evitando así la transcripción), y ofrecer la opción de **mostrar la contraseña** de forma explícita mientras se está escribiendo.

The wireframe shows a top header labeled "Login". Below it is a field labeled "Escribe tu nombre de usuario" with an empty input box. Below that is a field labeled "Escribe la contraseña" with an empty input box. At the bottom is a dark button with the text "Pusa aquí para entrar".

Figura 4: Wireframe de la vista de la página de login

❖ Página principal (Home)

En la página principal se muestra la relación de todas las recetas disponibles en la aplicación.

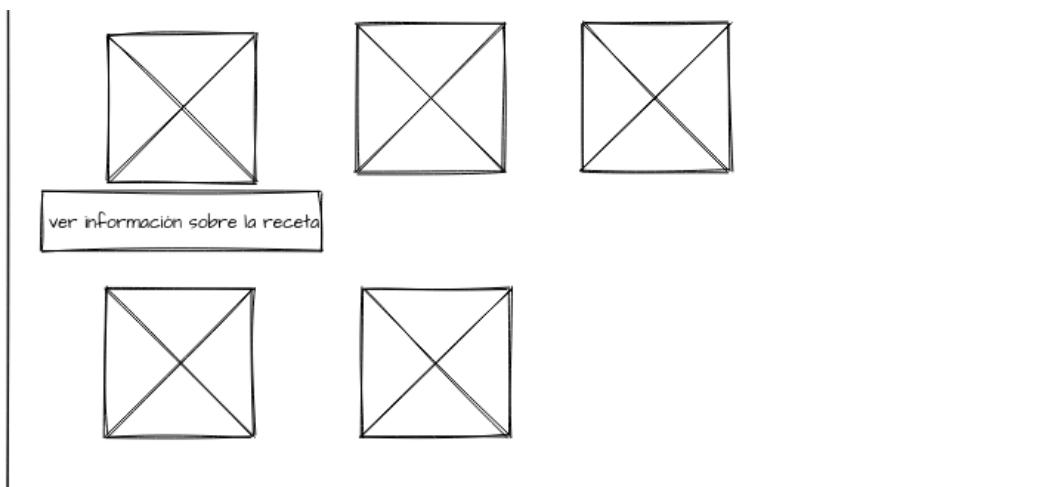


Figura 5: Wireframe de la vista de la página de principal donde se visualizan todas las recetas

El menú lo componen las acciones disponibles para cada grupo de usuario.

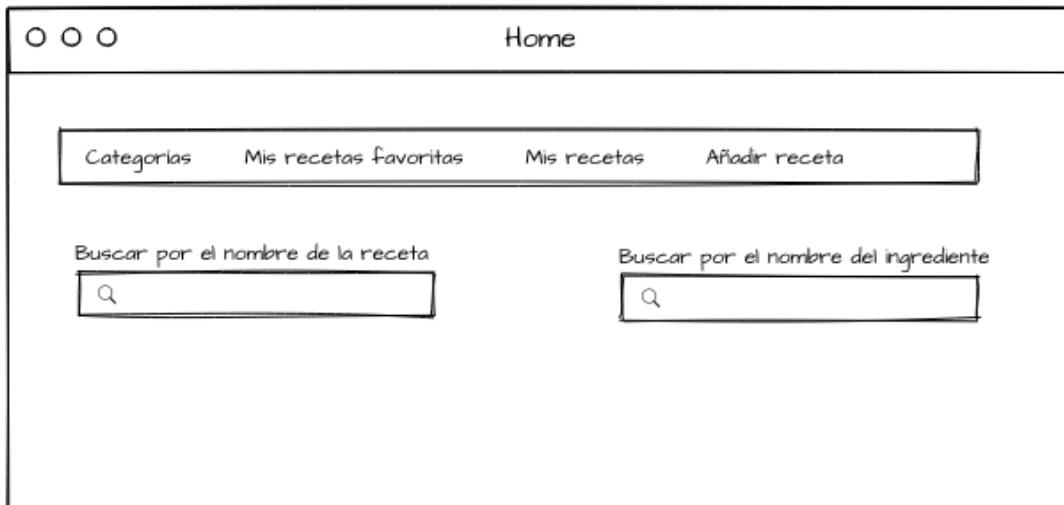


Figura 6: Wireframe del menú del perfil de usuario

The wireframe shows a header with three dots and the URL "/administrador". Below the header are three buttons: "Usuarios" (highlighted in black), "Añadir ingrediente", and "Añadir receta".

RECETAS	ACCIONES
USA	Editar-Eliminar
Sweden	Editar-Eliminar
Finland	Editar-Eliminar

INGREDIENTES	ACCIONES
USA	Editar-Eliminar
Sweden	Editar-Eliminar
Finland	Editar-Eliminar

Figura 7: Wireframe de la vista del menú del perfil de administrador

❖ Información sobre cada receta

Los pasos de las recetas están desglosados con el fin de facilitar su seguimiento, sobre todo para personas con problemas de memoria ejecutiva.

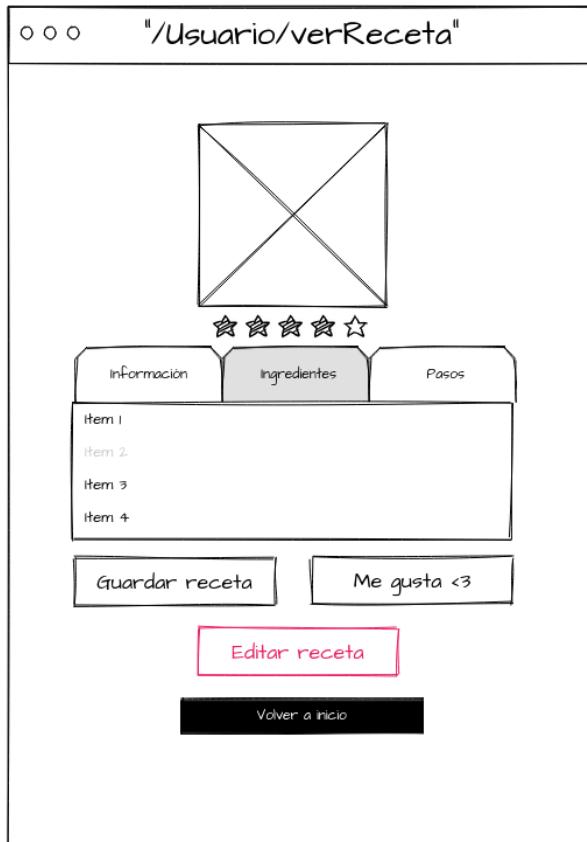


Figura 9: Wireframe de la vista de la receta

❖ Arquitectura de navegación

En cuanto a la arquitectura de navegación, los principales requisitos eran reducir el número de clics para realizar cualquier acción, especialmente la de ver una receta. En la siguiente imagen mostramos el diagrama de navegación para el usuario final.

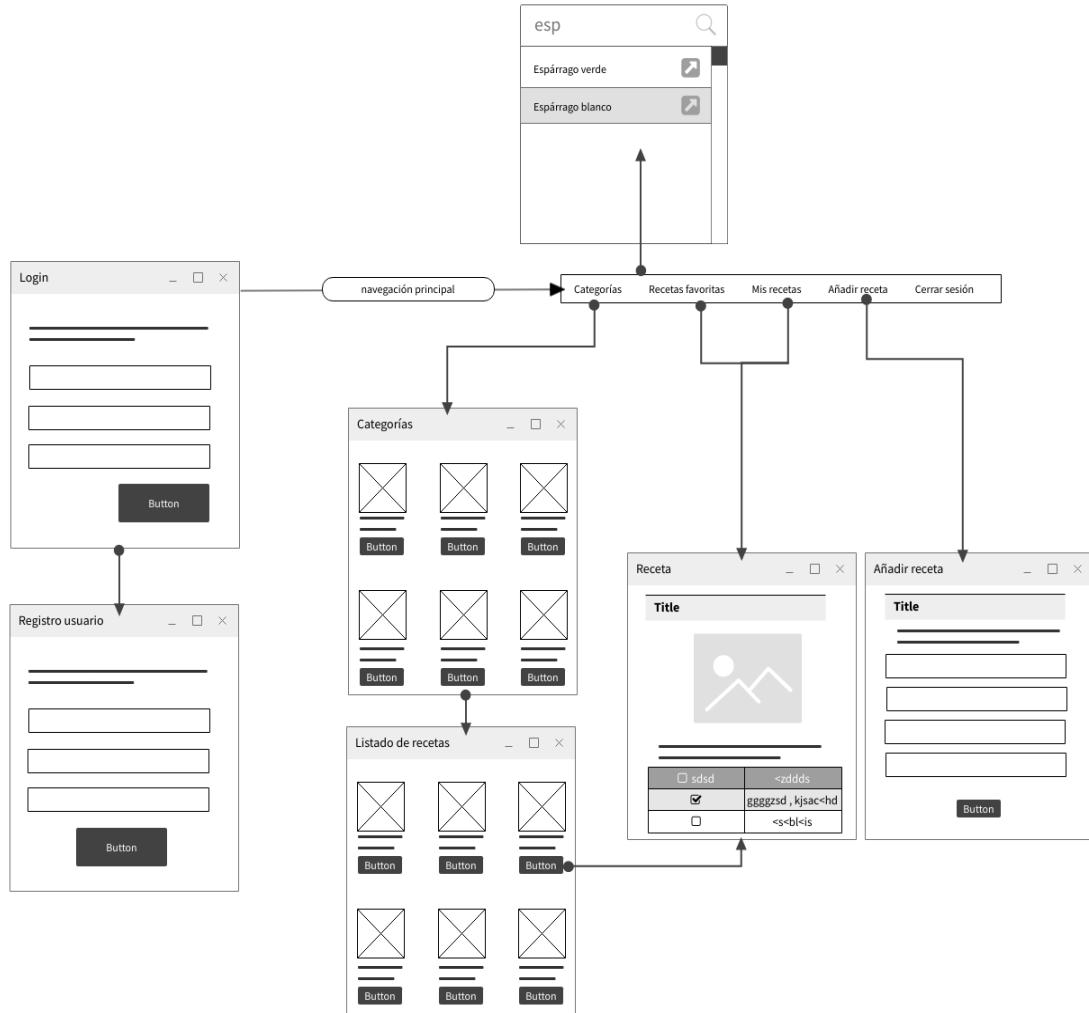


Figura 10: Arquitectura de navegación

Estilo

La accesibilidad cognitiva ha estado presente en todo momento, especialmente en la fase de diseño gráfico.

La tipografía utilizada en la realización del proyecto ha sido Open Sans, perteneciente a la familia Sans Serif. Esta familia de tipografías facilita la comprensión para los usuarios con dislexia, especialmente la primera de ellas. El ligero grosor de la parte inferior de las letras de OpenDyslexic ofrece una indicación de la orientación para que sea más difícil confundirlas con otras letras similares. Como fuentes alternativas empleamos Arial, Myriad Pro o Geneva, todas ellas son de la familia Sans Serif.

A continuación, se muestra el Look and Feel de la aplicación y las indicaciones de los usos admitidos para cumplir los estándares de accesibilidad AAA.



Figura 11: Logo y favicon



Figura 12: Guía de estilos

El color principal de la paleta empleada es un morado no agresivo a la vista que cuenta con un contraste alto sobre el fondo blanco. Los colores auxiliares son el lila y el amarillo que se emplean para elementos secundarios. El naranja, al ser un color intenso y brillante, se emplea en menor proporción para proporcionar instrucciones generales de navegación por la web. Para el fondo se ha prescindido de cualquier motivo o estampado puesto que puede ralentizar todavía más la lectura. Se han utilizado fondos sólidos para bloques de texto, principalmente de color blanco para optimizar el contraste.

La paleta de colores admite las siguientes combinaciones:

- Tipografía a 18pt o 14pt negrita y superior.
- Iconos y gráficos.

Por ejemplo: Header, logo, menú.

- Tipografía a 17pt e inferiores.
- Tipografía a 18pt o 14pt negrita y superior.
- Iconos y gráficos.

Por ejemplo: Títulos, párrafos, iconografía.

Por ejemplo:
Botones en estado normal
Y hoover.

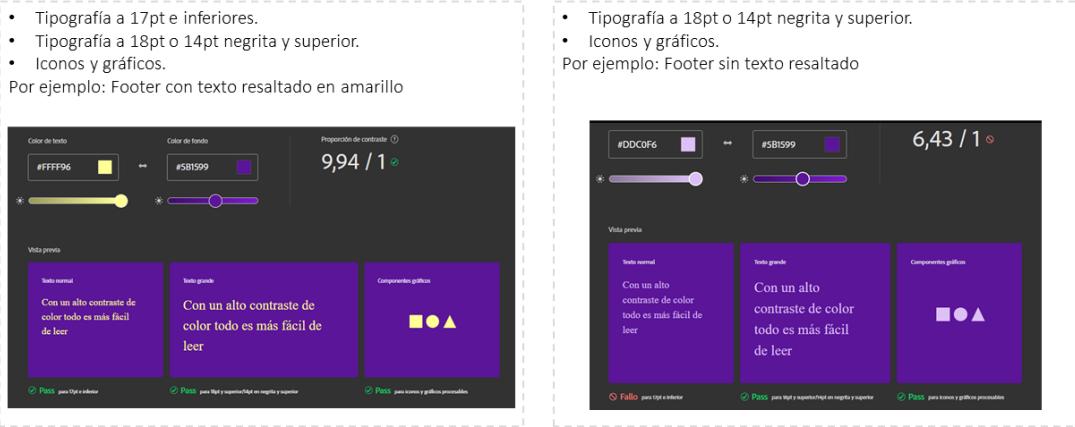


Figura 13: Combinaciones de colores aptas

Fase de desarrollo

Modelo de datos utilizado

La aplicación responde a una arquitectura de tres capas, en la que la interfaz de usuario (capa de presentación), la lógica de proceso (capa de negocio), el almacenamiento de datos y el acceso a ellos (capa de datos), se desarrollan en módulos independientes.

- **Capa de presentación:** Componente que interpreta las peticiones del usuario y presenta los resultados en el navegador web (cliente). El acceso a base de datos se realiza desde el modelo MVC (modelo vista controlador), con las vistas desarrolladas con ReactJS.
- **Capa de negocio:** Componente que mapea la base de datos en entidades y las pone al servicio de otros componentes, recibe las peticiones del usuario y envía las respuestas de dichas peticiones y verifica que se cumplen las reglas establecidas.
- **Capa de servicio:** Componente que se relaciona directamente con la base de datos y se sitúa entre la capa de negocio y la base de datos. En nuestra aplicación hemos configurado el acceso a la capa de datos con Spring Boot y el framework de Java JPA para crear la capa de persistencia. La principal ventaja de esta capa

es que conseguimos la flexibilidad para adaptarnos a cualquier gestor de base de datos. La capa de servicio (modelo DAO) llama al repositorio (modelo repository) para implementar los métodos definidos en las interfaces.

- **Capa de datos:** Componente que almacena, recupera y mantiene con integridad los datos. Para desarrollar esta capa hemos empleado MySQL Workbench.

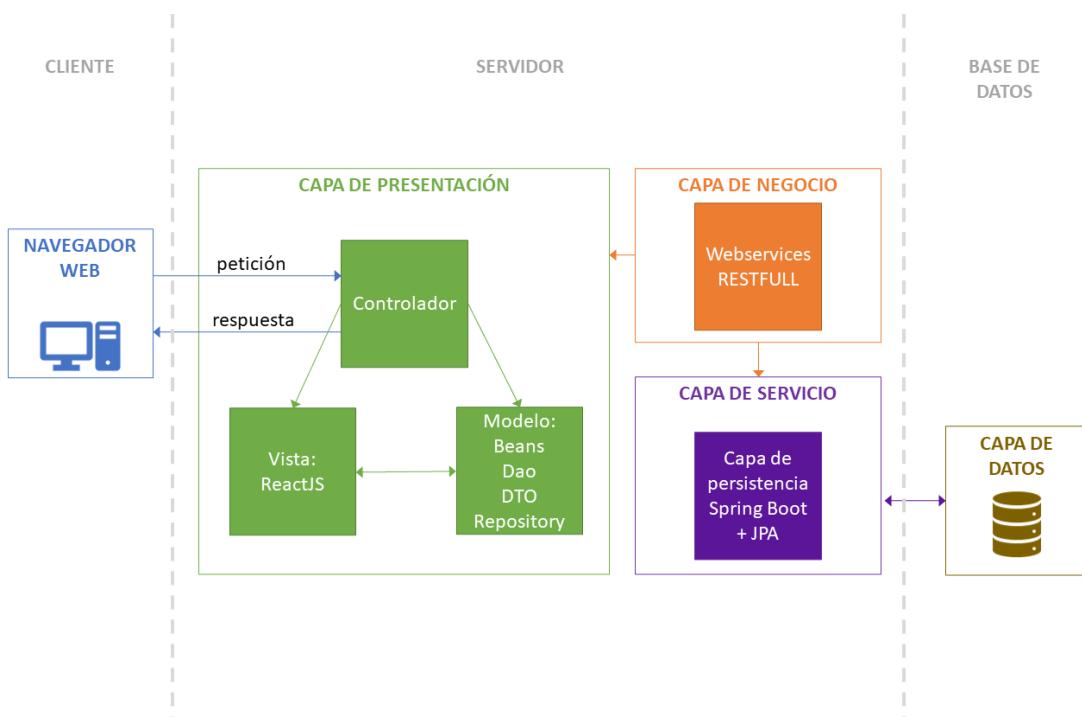


Figura 14: Esquema de la arquitectura de la aplicación

Diseño de la base de datos: diagrama de clases

En el siguiente modelo entidad-relación se ilustra cómo se relacionan entre sí las entidades dentro del sistema. Nótese que en la base de datos existen tablas que no se han empleado para el desarrollo de la versión beta estando disponibles para escalar las funcionalidades de la aplicación.

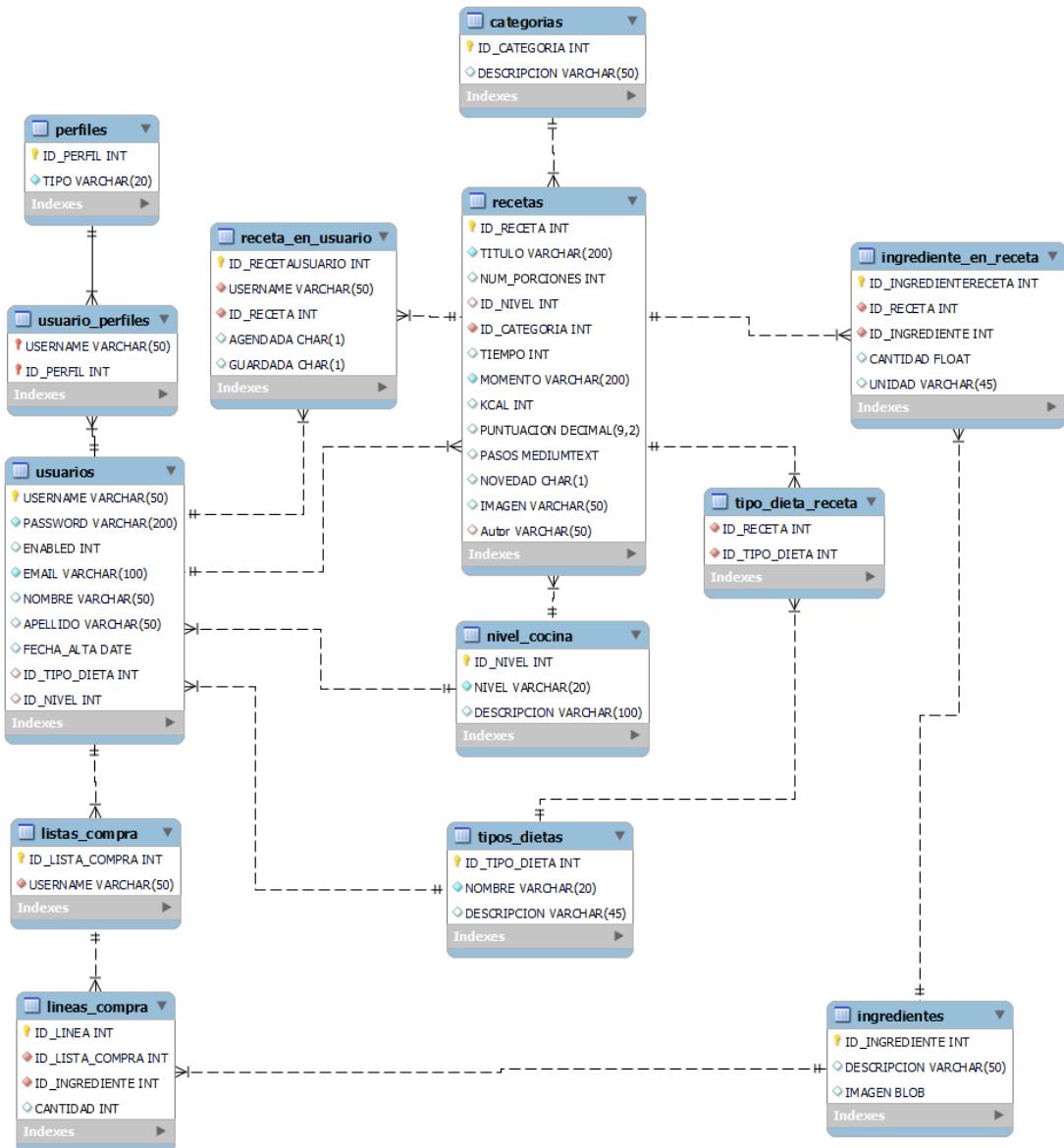


Figura 15: Diagrama de clases

Back-end

Como se ha comentado anteriormente, el back-end ha sido desarrollado usando el framework de Java, *SpringBoot*. La decisión de desarrollar una API Rest ha venido motivada por la tecnología usada en el front-end (*React JS*), ya que este no puede implementarse directamente en *SpringBoot*.

Su estructura general es la siguiente:

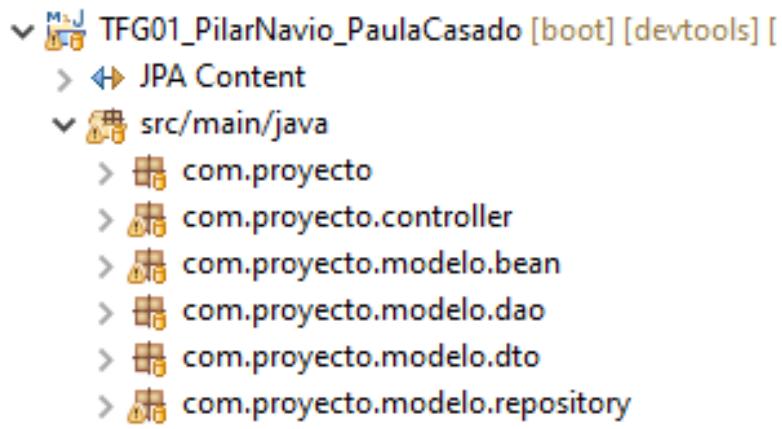


Figura 16: Estructura del back-end

Esta estructura surge como una buena práctica para realizar la conexión con la base de datos, implementar métodos para manejar los datos obtenidos y mostrar los resultados de esa manipulación. Para traducir entre los dos formatos de datos, de registros a objetos y de objetos a registros, se ha utilizado un **motor de persistencia**. Cuando el programa quiere grabar un objeto llama al motor de persistencia, que traduce el objeto a registros y llama a la base de datos para que guarde estos registros. De la misma manera, cuando el programa quiere recuperar un objeto, la base de datos recupera los registros correspondientes, los cuales son traducidos en formato de objeto por el motor de persistencia. Los frameworks que se encargan de realizar esta adaptación son conocidos como ORM (Object-Relational Mapping), siendo *Hibernate* el implementado en este proyecto.

En primer lugar, se han creado las entidades correspondientes a las tablas de la base de datos. Se encuentran en el paquete *com.proyecto.modelo.bean*. Las entidades en JPA no son más que un objeto plano (POJO) que representan datos que se pueden conservar en la base de datos. Una entidad representa una tabla almacenada en una base de datos, y cada instancia de una entidad representa una fila en la tabla. Cada clase contiene atributos que se corresponden con las columnas de cada tabla, un constructor sin parámetros, getter y setters, y hashCode>equals.

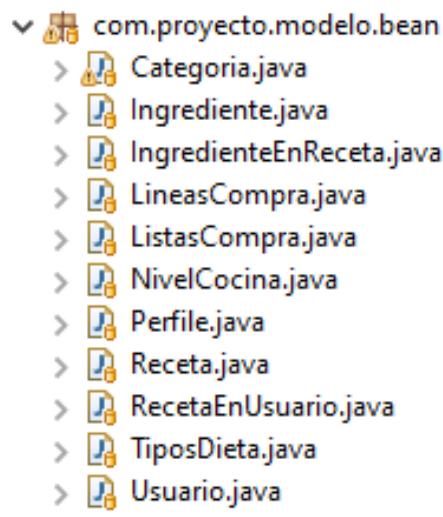


Figura 17: Clases Beans del modelo

El paquete *com.proyecto.modelo.repository* contiene interfaces que, a través de la implementación Jpa de Hibernate, proporciona tres clases con métodos para cada una de la mayoría de las operaciones CRUD. En este proyecto se ha usado la clase *JpaRepository*.

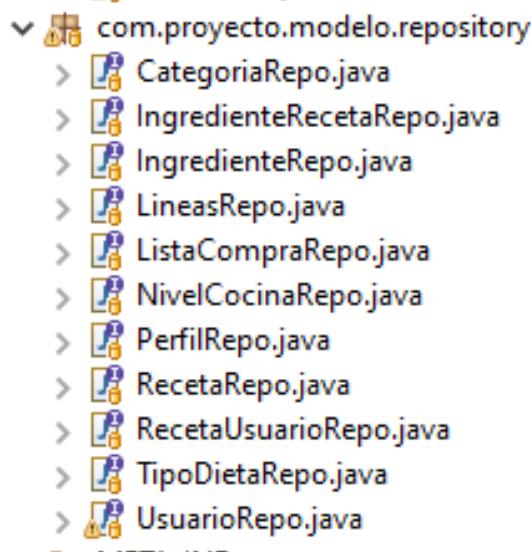


Figura 18: Interfaces que heredan los métodos de JpaRepository

Además de los métodos básicos, permite customizar queries de consulta a la base de datos:

```

public interface IngredienteRecetaRepo extends JpaRepository<IngredienteEnReceta, Integer>{

    @Query("select r from IngredienteEnReceta r where r.ingrediente.descripcion like %?1%")
    public List<IngredienteEnReceta> buscarXIngrediente(String descripcion);

    @Query("select ir from IngredienteEnReceta ir where ir.receta.titulo like %?1%")
    public List<IngredienteEnReceta> buscarXReceta(String titulo);
}

```

Figura 19: Muestra de las queries de consulta a base de datos implementadas en la interfaz del repositorio

Se ha dado el caso de necesitar métodos más específicos aparte de los aportados por JpaRepository para la manipulación de datos, por lo que se ha implementado un modelo DAO. Este sigue la misma lógica que el repositorio: encapsula toda la lógica de acceso de datos al resto de la aplicación.

Consisten en una interfaz con los métodos a implementar y una clase que implementa dicha interfaz. Los métodos harán uso tanto de los métodos básicos aportados por JpaRepository como los implementados en el repositorio usando los resultados de las queries customizadas.

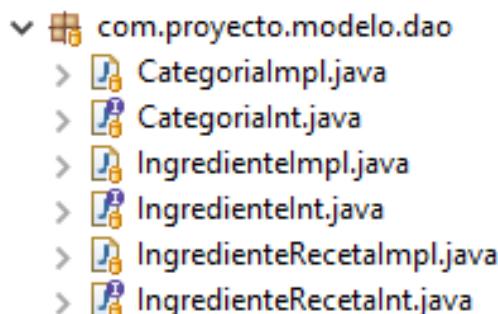


Figura 20: Interfaces y clases que implementan las interfaces del modelo DAO

Por otro lado, se ha creado un DTOs para la entidad *Receta*. El patrón DTO (Data Transfer Object) tiene como finalidad de crear un POJO con una serie de atributos que puedan ser enviados o recuperados del servidor en una sola invocación, de tal forma que un DTO puede contener información de múltiples fuentes o tablas y concentrarla en una única clase simple. Un DTO es una **buenas prácticas** global en la creación de la capa de negocio.

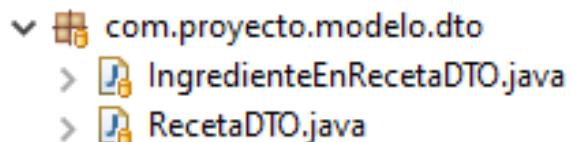


Figura 21: DTOs

Finalmente, la capa intermedia entre el modelo DAO y las vistas es el controlador. En este proyecto, se ha desarrollado un controlador común para los diferentes usuarios que van a hacer uso de la aplicación; uno para los clientes (*UsuarioRestController*) y otro para el administrador (*AdminController*), que contienen las funcionalidades que puede realizar cada rol.

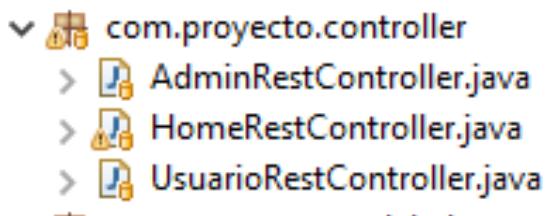


Figura 22: Clases del paquete Controller

Durante el desarrollo del back-end han surgido **tres problemas principales** que han dificultado el avance del proyecto. Por un lado, el diseño de la base de datos ha generado referencias cruzadas, provocando bucles infinitos en los resultados de las consultas realizadas desde el back-end. Para resolverlo, se recurre a la anotación de SpringBoot `@JsonIgnoreProperties`, la cual permite ignorar solo una o varias propiedades de la entidad cuando se declara con el modificador *value*, y no la entidad completa como ocurre con la anotación `@JsonIgnore`:

```

AdminController.java      Categoría.java      Ingrediente.java      TiposDieta.java      Receta.java
+o //bi-directional many-to-one association to IngredienteEnReceta
49@ OneToMany(mappedBy="receta")
50 private List<IngredienteEnReceta> ingredienteEnRecetas;
51
52 //bi-directional many-to-one association to RecetaEnUsuario
53@ OneToMany(mappedBy="receta")
54 private List<RecetaEnUsuario> recetaEnUsuarios;
55
56 //bi-directional many-to-one association to Categoría
57@ManyToOne
58 @JoinColumn(name="ID_CATEGORIA")
59 @JsonIgnoreProperties(value={"recetas"})
60 private Categoría categoria;
61
62 //bi-directional many-to-one association to NivelCocina
63@ManyToOne
64 @JoinColumn(name="ID_NIVEL")
65 @JsonIgnoreProperties(value={"usuarios", "recetas"})
66 private NivelCocina nivelCocina;

```

Figura 23: Muestra del uso de la anotación `@JsonIgnore`

Con esto se evitan los atributos de tipo lista que, a su vez, contienen otros objetos con atributos tipo lista.

En los casos en los que es necesario ignorar un atributo por completo, se opta por la anotación `@JsonIgnore`:

```

Controller.java      Categoría.java      Ingrediente.java      TiposDieta.java      Receta.java
+o
        )
    @JsonIgnoreProperties(value="usuarios")
    private List<Receta> recetas;

    //bi-directional many-to-one association to Usuario
    @OneToMany(mappedBy="tiposDieta")
    @JsonIgnore
    private List<Usuario> usuarios;

```

Figura 24: Muestra del uso de la anotación `@JsonIgnore`

Por otro lado, debido a las referencias cruzadas, no era posible insertar nuevos registros en determinadas tablas (concretamente, en *Ingredientes* y *Recetas*) mediante el método Post. Se encontraron dos posibles soluciones, una de ellas siguiendo buenas prácticas, pero se ha decidido implementar las dos en este proyecto para poder exemplificarlas:

- Enviar los atributos necesarios para la creación del objeto como parámetros usando la anotación `@RequestParam`. En el caso de que el atributo sea de

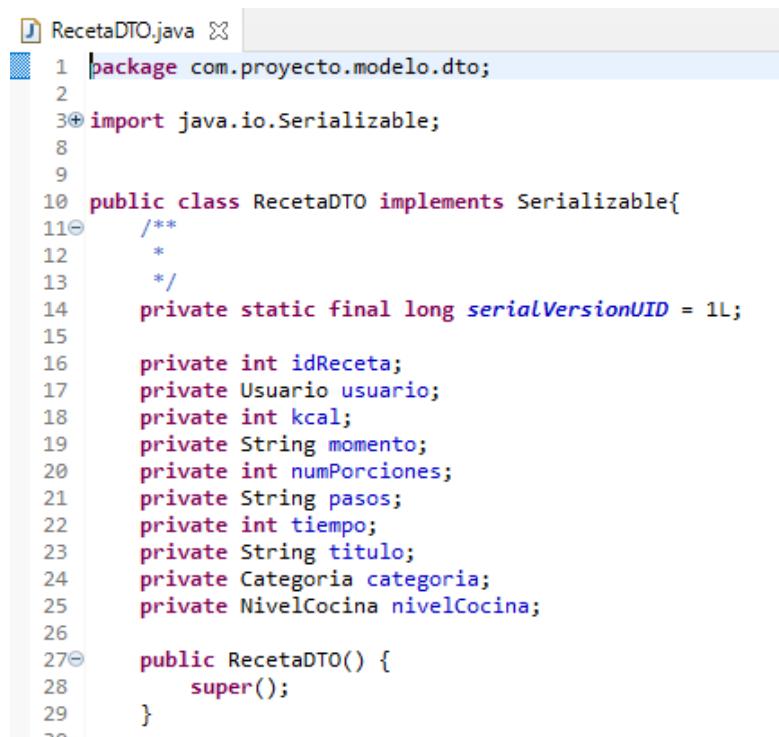
tipo primitivo, es necesario pasarlo como String y, posteriormente, aplicar un *cast* al tipo de dato deseado.

```
@PostMapping("/altaIngrediente")
public int registrarIngrediente(@RequestParam("descripcion") String descripcion) {
    System.out.println(descripcion);
    Ingrediente ingrediente=new Ingrediente();
    ingrediente.setDescripcion(descripcion);
    ingrediente.setIdIngrediente(3);
    if(idao.altaIngrediente(ingrediente)==1) {
        return 1;
    }else {
        return 0;
    }
}
```

Figura 25: Muestra del uso de la anotación @RequestParam

● Creación de un DTO para Receta.

1. Creación del DTO con los atributos necesarios para dar de alta una nueva receta:



```
RecetaDTO.java
package com.proyecto.modelo.dto;
import java.io.Serializable;
public class RecetaDTO implements Serializable{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private int idReceta;
    private Usuario usuario;
    private int kcal;
    private String momento;
    private int numPorciones;
    private String pasos;
    private int tiempo;
    private String titulo;
    private Categoría categoria;
    private NivelCocina nivelCocina;
    public RecetaDTO() {
        super();
    }
}
```

Figura 26: Fragmento de RecetaDTO. Atributos y constructor

2. Implementación del método “dar de alta” en el modelo DAO, que recibe un objeto del DTO creado. Se extraen los atributos para añadirlos a un

objeto Receta, el cual será finalmente insertado en la tabla correspondiente de la base de datos.



```

34             e.printStackTrace();
35
36         }
37         return filas;
38     }
39
40@*
41 public int altaReceta(RecetaDTO receta) {
42
43     Receta rec=new Receta();
44
45     rec.setCategoria(receta.getCategoría());
46     rec.setNivelCocina(receta.getNivelCocina());
47     rec.setMomento(receta.getMomento());
48     rec.setTitulo(receta.getTitulo());
49     rec.setNovedad("S");
50     rec.setKcal(receta.getKcal());
51     rec.setPasos(receta.getPasos());
52     rec.setTiempo(receta.getTiempo());
53     rec.setNumPorciones(receta.getNumPorciones());
54     rec.setUsuario(receta.getUsuario());
55     rec.setIdReceta(receta.getIdReceta());
56
57     int filas = 0;
58     try {
59         rrepo.save(rec);
60         filas = 1;
61     }catch(Exception e) {
62         e.printStackTrace();
63     }
64     return filas;
65 }
66

```

Figura 27: Método para dar de alta una receta recibiendo como parámetro un objeto de tipo RecetaDTO

3. Implementación del método “dar de alta” en el Controller. Recibe los atributos del DTO *RecetaDTO*:

```

@PostMapping("/altaReceta")
public String altaReceta(@RequestBody RecetaDTO receta, HttpSession session) {

    session.setAttribute("receta", null);
    Usuario usuario = (Usuario) session.getAttribute("usuario");

    receta.setUsuario(usuario);
    receta.setIdReceta(56);

    //Crea un objeto receta y la guarda en sesion
    Receta recetaSesion=rdao.recuperarSesion(receta);
    session.setAttribute("receta", recetaSesion);
    return(rdao.altaReceta(receta)==1)? "Si": "No";
}

```

Figura 28: Método para dar de alta una receta recibiendo como parámetro un objeto de tipo

RecetaDTO del Controller

Una vez se ha dado de alta una receta, es necesario incluir una lista de ingredientes y cantidades. Esto ha desembocado en el tercer problema surgido en este proyecto, ya que SpringBoot no permite enviar listas como parámetros de una url en el Controller. Ha sido necesario un cambio de diseño a la hora de crear una nueva receta con una lista de ingredientes, unidades y cantidades, de forma que en lugar de una lista de ingredientes, se añadirán los mismos uno a uno. El procedimiento es similar a la creación e inserción de un objeto Receta.

```
4
5 public class IngredienteEnRecetaDTO implements Serializable{
6
7     /**
8      *
9     */
10    private static final long serialVersionUID = 1L;
11
12    private int idIngrediente;
13    private float cantidad;
14    private String unidad;
15
16    public IngredienteEnRecetaDTO() {
17    }
18}
```

Figura 29: DTO de la clase IngredienteEnReceta

```
@Override
public int nuevaReceta (IngredienteEnReceta nuevaReceta) {
    try {
        irrepo.save(nuevaReceta);
    }catch(Exception e) {
        e.printStackTrace();
    }
    return 1;
}
```

Figura 30: Implementación del método que inserta un nuevo registro en la tabla IngredienteEnReceta.

En este caso, recibe un objeto IngredienteEnReceta en lugar un objeto del DTO.

```

@PostMapping("/añadirIngrediente")
public String altaIngredientes(@RequestBody IngredienteEnRecetaDTO nuevaReceta, HttpSession session) {
    IngredienteEnReceta recetaCreada=new IngredienteEnReceta();
    Ingrediente ingSeleccionado=idao.findById(nuevaReceta.getIdIngrediente());
    Receta buscada=(Receta)session.getAttribute("receta");
    recetaCreada.setCantidad(nuevaReceta.getCantidad());
    recetaCreada.setUnidad(nuevaReceta.getUnidad());
    recetaCreada.setIngrediente(ingSeleccionado);
    recetaCreada.setReceta(buscada);
    System.out.println(recetaCreada);
    return (irdao.nuevaReceta(recetaCreada)==1)?"Alta realizada":"Alta no realizada";
}

```

Figura 31: Implementación del método en el Controller. En este caso, recibe un objeto IngredienteEnRecetaDTO. Para poder asignar cada uno de los ingredientes a la receta creada anteriormente, se guarda la misma como atributo de sesión y se recupera posteriormente.

Para finalizar con los puntos importantes en el desarrollo del back-end, hay que destacar que se ha recurrido a la clase *ResponseEntity<T>* y la anotación *@ResponseStatus* para optimizar el manejo de las respuestas HTTP ante peticiones REST de Servicios. La clase ResponseEntity hereda de la clase HttpEntity y agrega un HttpStatus. Para los HttpStatus más populares, existen métodos estáticos de esta clase:

- BodyBuilder accepted();
- BodyBuilder badRequest();
- BodyBuilder created(java.net.URI location);
- HeadersBuilder<?> noContent();
- HeadersBuilder<?> notFound();
- BodyBuilder ok();

A continuación se muestran ejemplos de las diferentes formas en las que se han implementado estos recursos:

```

@GetMapping("/verUsuarios")
@ResponseStatus(HttpStatus.OK)
public List <Usuario> verUsuarios() {
    return udao.findAll();
}

```

Figura 32: Ejemplo de @ResponseStatus con código 200 (OK)

```

    @PostMapping ("/login")
    public ResponseEntity<Usuario> formLogin (HttpSession session, Usuario usuario) {
        Usuario usu=usuInt.login(usuario.getUsername(), usuario.getPassword());
        if (usu!=null) {
            session.setAttribute("usuario", usu);
        }else {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
        }
        return new ResponseEntity<Usuario>(usu, HttpStatus.OK);
    }

```

Figura 33: Ejemplo de `@ResponseEntity`. Devuelve un objeto `ResponseEntity` de tipo `Usuario` y el código 200 (OK) en el caso de que el login se haya realizado, o null y código 400 (Bad request).

Front-end

El front-end ha sido desarrollado en *React JS*, usando *Semantic UI React* como framework para agilizar el proceso de creación de interfaces. Esta fase del proyecto se ha solapado con el final de la fase de desarrollo del back-end, debido a que Spring Boot no permite implementar frameworks como React JS o Angular, lo que ha llevado a cambios continuos en el back-end a pesar de ser estrictamente independiente al front-end.

Sobre la estructura del front-end, hay que destacar que los componentes se han dividido en función de las vistas diseñadas y los roles de los usuarios, evitando anidaciones excesivas. En la siguiente imagen se muestra el esqueleto del proyecto en ReactJS:

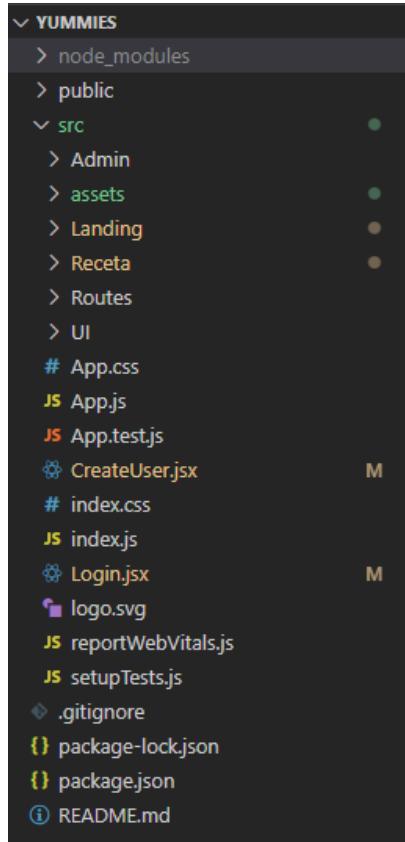


Figura 34: Estructura del front-end

El *login* y el *formulario para crear un nuevo usuario* suponen la vista principal de la aplicación y no están condicionadas por el rol del usuario, por lo que no se han incluido dentro de un componente.

En el componente *Admin* se encuentran las funcionalidades que puede realizar el rol de administrador, así como las vistas:

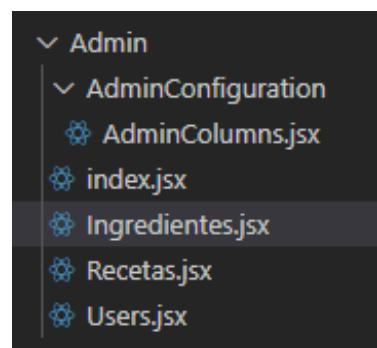


Figura 35: Estructura del componente Admin

El componente *assets* contiene los elementos estáticos (los estilos en CSS y las imágenes de las recetas):

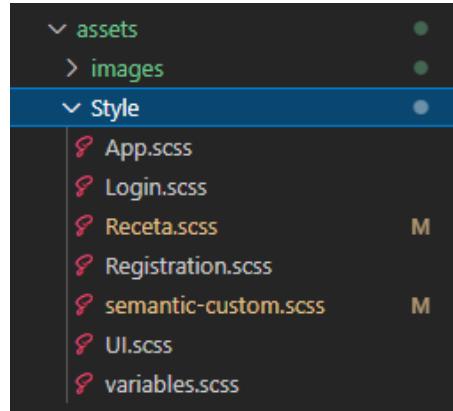


Figura 36: Estructura del componente Admin

El componente *Landing* constituye la vista de inicio, donde se muestran todas las recetas, filtros de búsqueda, opción de ver más información sobre una receta y el menú de navegación (mis recetas, recetas guardadas):

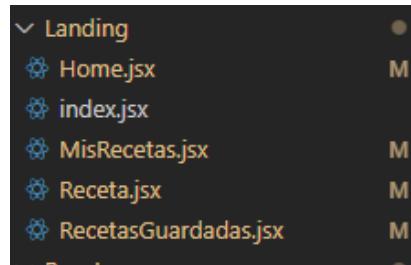


Figura 37: Estructura del componente Landing

El componente *Receta* se corresponde con el formulario para dar de alta una nueva receta y añadir ingredientes a la misma:



Figura 38: Estructura del componente Receta

El componente *Routes*, como indica su nombre, abarca las rutas para navegar en el front-end:

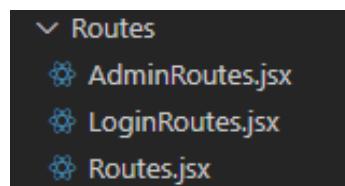


Figura 39: Estructura del componente Routes

```

export default function AriaRoutes () {
  return (
    <Routes>
      <Route path='/*' element={<Users />} />
      <Route path='/admin/*' element={<Users />} />
      <Route path='/admin/usuarios' element={<Users />} />
      <Route path='/admin/recetas' element={<Recetas />} />
      <Route path='/admin/ingredientes' element={<Ingredientes />} />
      <Route path='/addReceta' element={<AddReceta />} />
    </Routes>
  )
}

```

Figura 40: Rutas del rol administrador

```

export default function AriaRoutes () {
  return (
    <Routes>
      <Route path='/login' element={<Login />} />
      <Route path='/registration' element={<CreateUser />} />
      <Route path='/*' element={<Home />} />
    </Routes>
  )
}

```

Figura 41: Rutas del rol login y alta usuario

Finalmente, en el componente *UI* se han incluido elementos de diseño como la paginación de las tablas en las vistas de administrador.

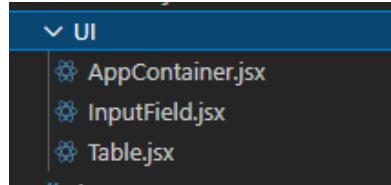


Figura 42: Elementos de diseño

Para realizar las peticiones al back-end, se ha utilizado Axios. Axios es una API HTTP de cliente basada en XMLHttpRequest, que se puede utilizar en el navegador y en un servidor con Node.js. Axios se basa en promesas, lo que le permite aprovechar `async` y `await` de JavaScript para obtener un código asíncrono más legible. También puede interceptar y cancelar solicitudes, y hay una protección integrada del lado del cliente contra la falsificación de solicitudes entre sitios. Axios es, en resumen, el recurso utilizado para hacer la conexión con el back-end:

```

const addIngrediente = _ => {
  setValidated(true)
  if(!checkEmpty()){
    const auxValues = {...values}
    auxValues.idReceta = idReceta
    console.log(auxValues)
    axios.post(`http://localhost:8088/rest/usuario/anadirIngrediente`, auxValues)
  }
}

```

Figura 43: Ejemplo de petición para añadir un ingrediente a la receta creada.

Aunque en la defensa de este proyecto se realizará una demo de la aplicación, se muestran a continuación capturas de algunos elementos que siguen estándares de accesibilidad cognitiva:

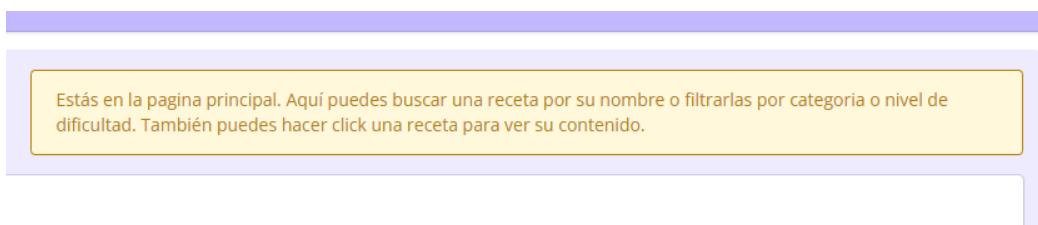


Figura 44. Las instrucciones de navegación de la vista se encuentran en un recuadro amarillo. Están presentes en las diferentes vistas de la aplicación.

Crear una receta

Para crear una receta, **rellena este formulario**. Cuando acabes, haz click en el botón del final de la página para continuar **añadiendo los ingredientes**.

Ponle un nombre (obligatorio)

Torrijas

¿En qué momento del día se puede comer? (obligatorio)

Ingrrese un momento para comerla

Figura 45. Input correcto en verde, incorrecto en rojo e instrucción para corregir el error. Esto es parte de la validación de formularios.

[Volver a las Recetas](#)



Torrijas

Figura 46. Botón para volver a la vista principal.

El desarrollo del front-end ha generado menos problemas que el back-end, ya que una de las integrantes del grupo ha recibido formación en React JS durante las prácticas (FCT), y se ha utilizado como apoyo un proyecto realizado durante las mismas. Esto ha sido crucial debido al escaso tiempo con el que se contaba para acabar esta fase.

Fase de pruebas

Test de Usabilidad

En la fase de pruebas se trata de medir la eficacia, eficiencia y la satisfacción a través de los tests de usuario de forma presencial para dar soporte en el caso de que el formulario no se entienda por completo y para pausar o posponer la prueba en el caso de que aparezcan signos de frustración o angustia en los participantes.

A los usuarios se les solicita que realicen varias tareas: buscar una receta para el postre, guardar una receta como favorita, dar de alta una receta.

Se cronometra el tiempo empleado en cada tarea para analizar posteriormente cuáles son en las que se invierte más tiempo. Esto puede darnos una idea de las partes en las que los usuarios encuentran dificultades.

En los tests de usabilidad también contemplamos el estado emocional que supone el uso de la aplicación y consultamos si se han frustrado o enfadado en algún momento.

Antes de la entrega de esta memoria solo se ha podido realizar el test de usabilidad a dos personas con perfil compatible a los usuarios objetivo. El número de participantes ha sido tan reducido que los resultados no son significativos, pero nos sirven de orientación para determinar que la parte más difícil de navegar es la del formulario para dar de alta una receta.

Test de Accesibilidad

Se ha aplicado la herramienta Wave para evaluar la accesibilidad de la página. Analizando los errores obtenidos, no se pueden aplicar estrictamente al diseño de la aplicación, ya que el principal error es la ausencia de etiquetas que, sin embargo, se encuentran presentes. En las siguientes imágenes se muestran los resultados de algunas vistas.

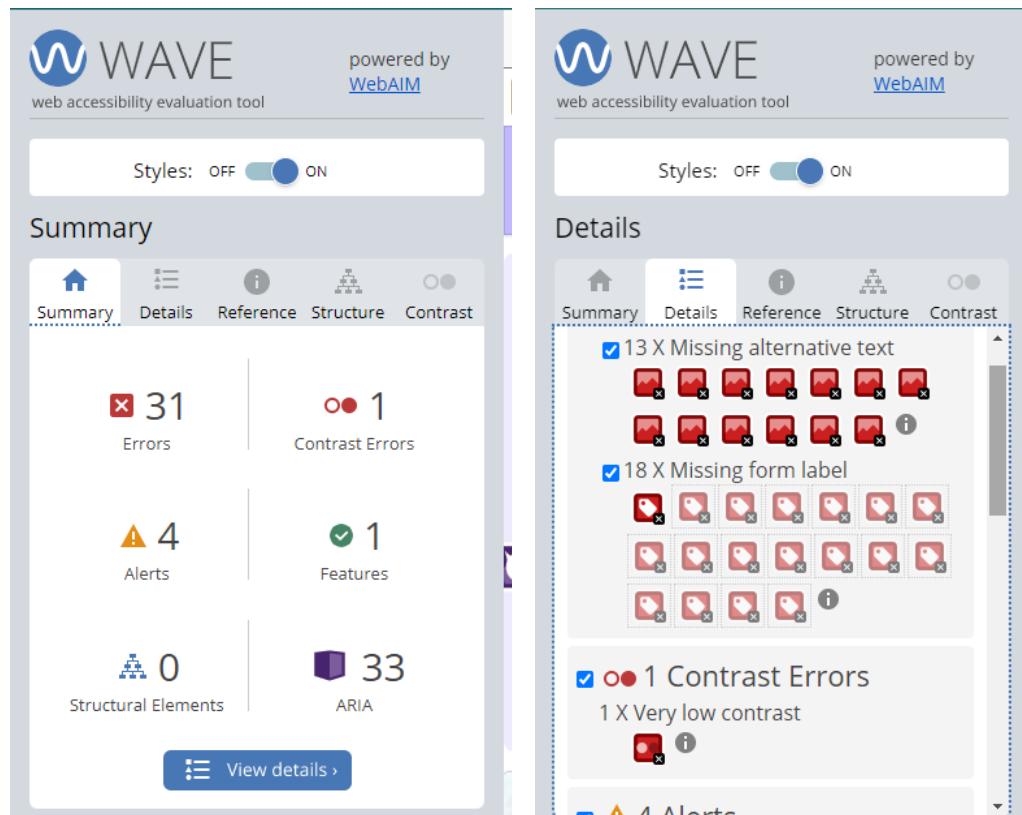


Figura 47: Página principal: muestra todas las recetas y los filtros

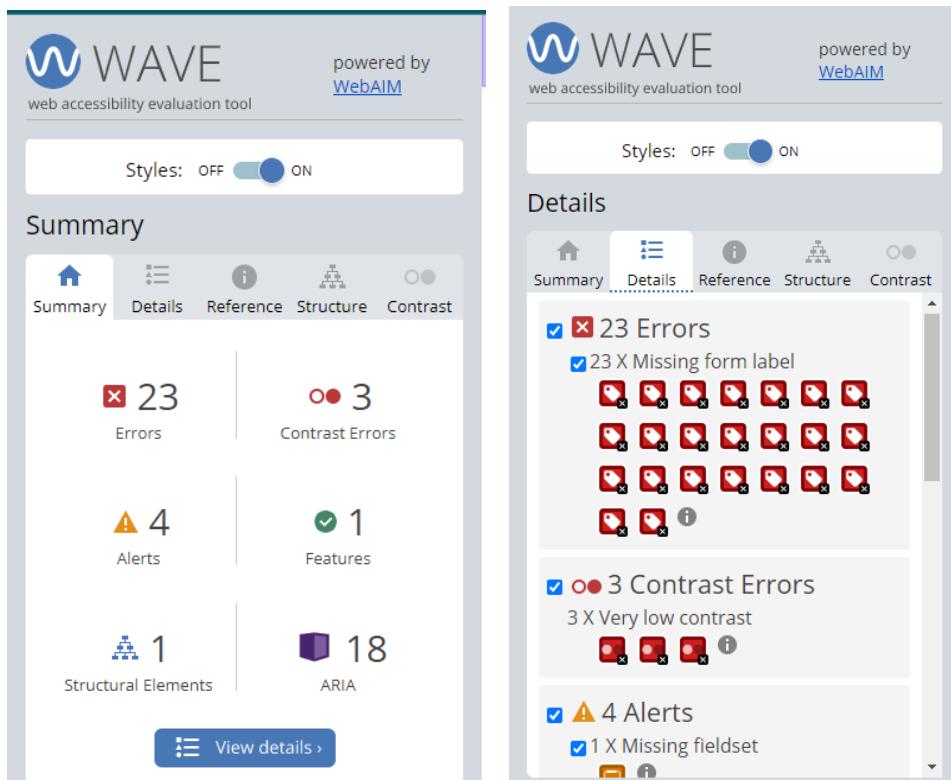


Figura 48: Formulario para dar de alta una receta

Los principales errores corresponden a las labels de las imágenes. Deben de revisarse estas vulnerabilidades antes de subir el sitio web a un entorno de producción.

Fase de lanzamiento

La fase de lanzamiento debería haber consistido en el despliegue de la aplicación en un servidor remoto, pero finalmente no se ha efectuado.

Se trata de una aplicación local, y para ejecutar la aplicación en el propio equipo es necesario importar el zip en Eclipse y ejecutar el script de la base de datos.

Las tecnologías necesarias para ejecutar el proyecto son:

- El gestor de bases de datos MySQL Workbench
- Framework SpringBoot (back-end)
- Framework React JS (front-end)
- Un IDE que soporte ambas tecnologías. En este proyecto se han utilizado Eclipse for Java EE Developers (back-end) y Visual Studio Code (front-end)

Para más información puede consultarse en el anexo la guía de usuario así como la presentación del proyecto.

Conclusiones y mejoras del proyecto

Conclusiones

En este proyecto hemos intentado plasmar los conocimientos adquiridos a lo largo de estos dos años de formación y desarrollar de forma holística aquellas habilidades que hasta ahora habíamos trabajado independientemente en cada módulo formativo.

Además, hemos querido integrar una pequeña parte de nuestra formación de base, la neuropsicología aplicada a personas con diversidad funcional, creando una aplicación web que podría facilitar la inclusión de personas con disfunción cognitiva. Nuestra experiencia así como la investigación documental llevada a cabo fueron los principales cimientos para determinar cuáles iban a ser los requisitos funcionales de la aplicación.

El diseño del código fuente y de la base de datos han seguido una metodología “database first”, puesto que el diseño de la base de datos ha condicionado el diseño del resto de componentes.

A lo largo del proceso de desarrollo nos hemos encontrado con situaciones que han dificultado seguir la planificación del anteproyecto. La principal dificultad ha sido decantarnos por la tecnología utilizada para desarrollar el front-end. Desde el primer momento intentamos hacerlo con Angular, pero su curva de aprendizaje estaba consumiendo más tiempo del esperado, por lo que tuvimos que decantarnos por una de estas opciones: hacer vistas con JSP modificando el back-end o usar otra tecnología de frontend que se adaptara a nuestras necesidades, como React JS. Finalmente, preferimos esta última opción para aprovechar el back-end desarrollado.

Líneas de futuro

Las dificultades sobrevenidas codificando el back-end han condicionado en gran medida el avance del proyecto y han obligado a reducir los casos de uso originalmente planteados. No obstante, se establecen como objetivos de futuro para desarrollar en versiones posteriores.

La base de datos se ha diseñado para dar cabida a otras funcionalidades para un tipo de usuario premium. Se propone hacer una pasarela de pago para dar las credenciales para estas funcionalidades:

- Valorar la receta.
- Agendar la receta en el calendario de Google Calendar haciendo uso de la API de Google: <https://developers.google.com/calendar/api/downloads>.
- Añadir a la lista de la compra una receta: se añaden: nombre de la receta, ingredientes.
- Ver la lista de la compra con los ingredientes que se necesitan para las recetas agendadas.
- Recetas sugeridas: Listado de recetas teniendo en cuenta las preferencias del usuario: nivel de cocina y tipo de dieta.
- Marcar las alergias e intolerancias y evitar la visualización de recetas que contengan algún ingrediente incompatible.

Para el desarrollo en el que intervienen varios tipos de usuario se propone el uso de Spring Security.

En cuanto a las funcionalidades del front-end que no se han podido desarrollar en este primer evolutivo señalamos:

- Acompañar el nombre de cada ingrediente con su imagen.



Figura 49: Wireframe de los listados de ingredientes

- Las medidas de las cantidades de los ingredientes estaban concebidas para que se mostrasen en formato escrito en lugar de numérico junto con unidades de medida familiares para el usuario. Por ejemplo: una cucharada, en lugar de 10ml. La solución propuesta consistía en mapear las cantidades numéricas con su correspondiente grafía y modificar el atributo de cantidad con JavaScript.

- Actualizar las cantidades de los ingredientes en función del número de comensales. En la base de datos la cantidad de los ingredientes viene establecido para una ración, por lo que la solución sería multiplicar el valor del atributo del número de comensales a la cantidad de cada ingrediente en receta y mapearlo a su grafía correspondiente como se ha explicado en el párrafo anterior.

Bibliografía

Nota: la bibliografía sigue las normas de formato de la American Psychological Association (APA).

Aggarwal, S. (2019, 23 agosto). *CrudRepository, JpaRepository, and PagingAndSortingRepository in Spring Data*. Baeldung. Recuperado 17 de mayo de 2022, de <https://www.baeldung.com/spring-data-repositories>

Baeldung. (2022a, enero 9). *Spring @RequestParam Annotation*. Recuperado 17 de mayo de 2022, de <https://www.baeldung.com/spring-request-param>

Baeldung. (2022b, 31 marzo). *The DTO Pattern (Data Transfer Object)*. Baeldung. Recuperado 17 de mayo de 2022, de <https://www.baeldung.com/java-dto-pattern>

British Dyslexia Association. (2018). *Dyslexia friendly style guide*. Recuperado 17 de mayo de 2022, de <https://www.bdadyslexia.org.uk/advice/employers/creating-a-dyslexia-friendly-workplace/dyslexia-friendly-style-guide>

Carnegie Museums of Pittsburgh Innovation Studio. (s. f.). *Forms | Accessibility Guidelines*. STUDIO. Recuperado 17 de mayo de 2022, de <http://web-accessibility.carnegiemuseums.org/code/forms/>

Filatova, O. (s. f.). *Bonbon Style Vector Illustrations in PNG and SVG*. Icons8. Design Tools. Recuperado 17 de mayo de 2022, de <https://static-cdn.icons8.com/illustrations/style--bonbon>

Gierke, O., Strobl, C., Paluch, M., Krabbenborg, S., Wouters, J., & Turnquist, G. (2022, 13 mayo). *JpaRepository (Spring Data JPA 2.7.0 API)*. Spring Data JPA. Recuperado 17 de mayo de 2022, de <https://docs.spring.io/spring-data/data-jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html>

Introduction - Semantic UI React. (s. f.). Semantic UI React. Recuperado 17 de mayo de 2022, de <https://react.semantic-ui.com/>

Jongsiriyanyong, S., & Limpawattana, P. (2018). Mild Cognitive Impairment in Clinical Practice: A Review Article. *American Journal of Alzheimer's Disease & Other Dementiasr*, 33(8), 500–507. <https://doi.org/10.1177/1533317518791401>

JsonIgnoreProperties (Jackson-annotations 2.6.0 API). (2015). Annotation Type JsonIgnoreProperties. Recuperado 17 de mayo de 2022, de <https://fasterxml.github.io/jackson-annotations/javadoc/2.6/com/fasterxml/jackson/annotation/JsonIgnoreProperties.html>

Kirova, A. M., Bays, R. B., & Lagalwar, S. (2015). Working Memory and Executive Function Decline across Normal Aging, Mild Cognitive Impairment, and Alzheimer's Disease. *BioMed Research International*, 2015, 1–9. <https://doi.org/10.1155/2015/748212>

Manera, V., Petit, P. D., Derreumaux, A., Orvieto, I., Romagnoli, M., Lyttle, G., David, R., & Robert, P. H. (2015). 'Kitchen and cooking,' a serious game for mild cognitive impairment and Alzheimer's disease: a pilot study. *Frontiers in Aging Neuroscience*, 7. <https://doi.org/10.3389/fnagi.2015.00024>

Ophoff, J., Johnson, G., & Renaud, K. (2021). Cognitive function vs. accessible authentication: insights from dyslexia research. *Proceedings of the 18th International Web for All Conference*, 1–5. <https://doi.org/10.1145/3430263.3452427>

Plena Inclusión Madrid. (2020, noviembre). *Pautas de accesibilidad cognitiva web*. <https://plenainclusionmadrid.org/wp-content/uploads/2020/12/Guia-Pautas-Accesibilidad-2020-final.pdf>

ResponseEntity (Spring Framework 5.3.20 API). (s. f.). Spring Framework. Recuperado 17 de mayo de 2022, de <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/ResponseEntity.html>

Saunders, N. L. J., & Summers, M. J. (2009). Attention and working memory deficits in mild cognitive impairment. *Journal of Clinical and Experimental Neuropsychology*, 32(4), 350–357. <https://doi.org/10.1080/13803390903042379>

Web Accessibility Evaluation Tool (WAVE) <https://wave.webaim.org/>

W3C. (2021, April). *Making Content Usable for People with Cognitive and Learning Disabilities*. Recuperado 17 de mayo de 2022, de <https://www.w3.org/TR/coga-usable/>

W3C. (2022, May). *Provide a Login that Does Not Rely on Memory or Other Cognitive Skills*. Web Accessibility Initiative (WAI). Recuperado 17 de mayo de 2022, de <https://www.w3.org/WAI/WCAG2/supplemental/patterns/o6p01-login-cognition/>

Walden, A. (2018, 8 febrero). *Accessible Form Validation - Alison Walden*. Medium. Recuperado 17 de mayo de 2022, de <https://lsnrae.medium.com/accessible-form-validation-9fa637ddb0fc>

Anexos

Contenido del proyecto

Este proyecto consta de:

- BBDD: contiene el script de la base de datos para crear las tablas y algunos registros, con el fin de poder probar posteriormente la aplicación.
- TFG01_PilarNavio_PaulaCasado: back-end desarrollado en Java usando el framework SpringBoot.
- Resources: recursos utilizados como guía de estilo (look and feel).
- yummies: front-end desarrollado en React JS.
- Memoria TFG: documento formal del proyecto.
- ExposiciónPowerPoint: presentación utilizada en la defensa del proyecto. Formato Power Point.

El **back-end** utiliza el puerto **8088**, al cual se realizan las llamadas desde el front-end. El **front-end** de este proyecto, como se ha comentado, ha sido desarrollado en React JS, por lo que se establece automáticamente el puerto **3000**.

Repositorio de Github

El proyecto se encuentra alojado en Github: <https://github.com/PaulaCsdo/TFG-DAW>

Guía de usuario

Se va a realizar un **tutorial de uso de la aplicación** mediante un **vídeo**. Al ser un objetivo secundario, este se presentará como alternativa en el caso de que ocurra algún error durante la presentación de la demo en directo durante la defensa y será subido a Github posteriormente a la entrega de la memoria el día 20 de mayo.