universidad de león

Escuela de Ingenierías Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo de Fin de Grado

ZUCCA - APLICACIÓN WEB DE RECETAS CULINARIAS CON MERN STACK

ZUCCA - WEB APPLICATION FOR COOKING RECIPES USING MERN STACK

Autora: María Álvarez Castelo
Tutora: Lidia Sánchez González
Cotutor: Jairo Luzón Hernández

Enero, 2022

UNIVERSIDAD DE LEÓN
Escuela de Ingenierías I.I.
GRADO EN INGENIERÍA INFORMÁTICA
Trabajo de Fin de Grado

ALUMNO: María Álvarez Castelo

TUTOR: Lidia Sánchez González

TÍTULO: Zucca - Aplicación web de recetas culinarias con MERN stack

TITLE: Zucca - Web application for cooking recipes using MERN stack

CONVOCATORIA: Junio, 2022

RESUMEN:

Debido a la digitalización que ha habido en los últimos años de las tareas cotidianas, se demandan aplicaciones que faciliten el día a día de las personas. El proyecto describe el desarrollo de una arquitectura basada en microservicios que se puede aplicar para la creación de diferentes aplicaciones culinarias. Para ello, se implementa un sistema de recopilación y almacenamiento de datos empleando Kafka como medio de transmisión, y Docker como herramienta para su despliegue. En la página web denominada Zucca y creada a partir de este sistema, los usuarios pueden realizar búsquedas de la receta que quieran cocinar, filtrar por categorías, crearse una cuenta y guardar para consultar más tarde las recetas deseadas. Desarrollada con la pila de desarrollo MERN, se utiliza MongoDB como sistema de almacenamiento, Express.js como framework de Node.js para desarrollo de aplicaciones web, React.js para el desarrollo de las interfaces, y Node.js para el desarrollo del lado del servidor. Las pruebas diseñadas y aplicadas al producto final validan el correcto funcionamiento del sistema, cumpliendo con los objetivos planteados en este proyecto.

ABSTRACT:

Due to the digitalization of everyday tasks in recent years, there is a demand for applications that make people's daily lives easier. The project describes the development of an architecture based on microservices that can be applied to the creation of different culinary applications. For this purpose, a data collection and storage system is implemented using Kafka as a transmission medium, and Docker as a tool for its deployment. In the web page called Zucca and created from this system, users can search for the recipe they want to cook, filter by categories, create an account and save to consult the desired recipes later. Implemented with the MERN development stack, it uses MongoDB as the storage system, Express.js as the Node.js framework for web application development, React.js for interface development, and Node.js for server-side development. The tests designed and applied to the final product validate the correct operation of the system, fulfilling the objectives set out in this project.

Palabras clave: Aplicación web, Kafka, MERN, recetas, microservicios, página web	
Firma del alumno:	**VºBº Tutor:**

Índice de contenidos

Índice de figuras

Índice de tablas

Glosario de términos

- **API (Application Programming Interface):** conjunto de definiciones y protocolos empleados para desarrollar e integrar el software de aplicaciones, permitiendo la comunicación entre ellas a través de solicitudes y respuestas [1].
- **API REST:** interfaz de programación que permite la interacción con los servicios web de RESTful. Las solicitudes y respuestas se entregan por medio de HTTP en formatos como JSON [2].
- **Broker:** en una arquitectura Kafka, se puede definir como un servidor. Contiene particiones de diferentes *topics*, y, el tener varios *brokers*, hace que los datos que conforman un *topic* permanezcan seguros ante fallos y escalables [3].
- **Clúster:** en una arquitectura Kafka, un clúster es un conjunto de servidores (brokers) encargado de grabar los datos de los *topics* y particiones. Permite la división del trabajo y de los datos para balancear la carga del sistema [4] [3].
- **Docker compose:** herramienta de Docker que permite definir y ejecutar varios contenedores a través de un archivo YAML de configuración. Permite el despliegue de estos contenedores con un solo comando [5].
- **Framework:** [6]
- **Hub:** en el ámbito de contenido web, se define como un centro de contenido que ofrece recursos sobre un tema determinado [7].
- **Indeed:** página web que permite buscar los salarios medios de ciertos empleos según la parte de España [8].
- **Kafka consumer:** en una arquitectura Kafka, es el consumidor de los datos publicados por el Producer. Para consumir los datos debe estar suscrito al tema [9].
- **Kafka producer:** en una arquitectura Kafka, es el encargado de generar y publicar datos a un tema (topic) determinado [10].
- **Kafka:** sistema de modelo publicación / suscripción que permite la captura y transmisión de datos de diferentes fuentes de eventos [11].

- **Microservicios:** son los servicios, plataformas y componentes independientes que los desarrolladores emplean para implementar el ciclo de vida del desarrollo de software. Gracias a esta independencia de componentes se permite el uso de ciertos microservicios en otros proyectos [12].
- **Modelo publicación/suscripción:** modelo de comunicación formado por un grupo que genera la información y la publica en un determinado tema, y otro que la consume. Para poder consumir estos datos los consumidores deben estar suscritos a ese tema [13].
- **Pruebas de validación alfa:** pruebas realizadas por el cliente en el lugar de desarrollo utilizando el software de manera natural. El equipo de desarrollo, calidad o pruebas está presente como observador del usuario [14].
- **Responsive:** el diseño web responsive es aquel diseño que es capaz de adaptarse correctamente a pantallas de diferentes tamaños [15].
- **Topic:** en una arquitectura Kafka, un *topic* es un flujo de datos sobre un tema en particular. Estos *topics* almacenan los mensajes que son enviados por el producer para su posterior lectura por el/los consumer/s [3].
- **ZooKeeper:** servicio centralizado necesario para el funcionamiento de Kafka. Mantiene una lista de metadatos y mecanismos para gestionar los brokers, además de enviar notificaciones a Kafka ante cambios de *topics* o brokers [3].

0 Introducción

Hoy día, en Internet, existen multitud de páginas web que ofrecen recetas de cocina. Una gran parte de ellas no cuentan con un diseño *responsive* y accesible, y tampoco una navegación ágil e intuitiva. El proyecto que se presenta a continuación, el cual está formado por microservicios, describe el proceso de desarrollo de una arquitectura que realiza todo el flujo desde la recopilación de datos externos hasta su visualización en una página web. Se proporciona una aplicación web como una posible solución de las muchas que se podrían generar con la estructura Kafka, la cual solicita datos a una API externa y los almacena en una base de datos. En la aplicación web, los usuarios podrán descubrir desde cualquier dispositivo recetas culinarias y realizar búsquedas, así como explorar por categorías y marcar como favoritas las recetas deseadas.

0.1 PLANTEAMIENTO DEL PROBLEMA

Con el avance del uso de la tecnología, la mayoría de las situaciones cotidianas que realizamos a diario se han digitalizado. El hecho de consultar recetas culinarias, a diferencia de antiguamente, ya reducidas veces se realiza mediante medios físicos como pueden ser libros o cuadernos. La sociedad se adapta a estos cambios, creciendo el uso de Internet para tareas como consultar recetas personalizadas, obtener ideas de qué cocinar con ciertos ingredientes, guardarlas para consultarlas más tarde, compartirlas, e incluso crearlas. Mediante la implementación de una arquitectura capaz de recopilar datos automáticamente de una API REST externa y almacenarlos en un sistema de almacenamiento, se obtiene una solución a partir de la cual utilizar los datos recogidos para crear diferentes tipos de aplicaciones [2]. En ellas, los usuarios de Internet podrán consultar recetas, o realizar cualquier otro tipo de acción relacionada con estos datos.

0.2 OBJETIVOS

El objetivo del proyecto es desplegar un sistema basado en microservicios capaz de realizar todo el flujo desde la recopilación de los datos externos hasta su visualización en una página web. Gracias a la arquitectura de recopilación y almacenamiento, se prevé ofrecer esta solución de forma que cualquier desarrollador pueda utilizarla para crear diferentes aplicaciones web de recetas culinarias.

En este proyecto, se implementa una aplicación web a partir de la arquitectura mencionada anteriormente. Accesible para todos los usuarios, permite fácilmente obtener recetas, realizar búsquedas de las mismas y filtrarlas por categorías. Estos usuarios podrán darse de alta en la aplicación y realizar tareas como marcar las recetas como favoritas para consultarlas más tarde, así como eliminarlas.

También, se pretende obtener conocimientos sobre las tecnologías en las que se conforma la aplicación, ya que con algunas de ellas no se está familiarizado.

Para conseguir esto, se han determinado los siguientes subobjetivos:

- Analizar las tecnologías disponibles en el ámbito de recolección, transmisión, almacenamiento de datos e implementación de páginas web dinámicas.
- Establecer los requisitos funcionales y no funcionales, de cara a implementar el sistema.
- Realizar prototipos del diseño tanto de la arquitectura como de las interfaces de la página web.
- Desplegar la arquitectura para disponer de un sistema de almacenamiento y uno de peticiones.
- Implementar el desarrollo de la lógica de la aplicación, y su diseño web.
- Evaluar los resultados obtenidos y el cumplimiento de los requisitos.
- Elaborar un manual de usuario para el despliegue del sistema y el uso de la aplicación web.
- Realizar la documentación correspondiente a todo el proceso.

0.3 METODOLOGÍA EMPLEADA

El modelo de trabajo escogido se basa en metodologías ágiles, que siguen un desarrollo iterativo e incremental. Este tipo de metodología destaca por cubrir las debilidades de un modelo tradicional, los cuales suelen implicar un desarrollo más lento y que se adapta difícilmente a los cambios.

Gracias al desarrollo iterativo e incremental, el trabajo se puede dividir en un proceso de iteraciones o fases (en este caso de tres semanas) en el que al final de cada iteración se cuenta con un producto que tiene más funcionalidad que en la fase anterior. Además, cada una de estas implica también hacer mejoras en la aplicación respecto a la anterior. Los objetivos tratados en cada iteración son subconjuntos de funcionalidades de nuestra aplicación final.

Para cada etapa, se realiza:

- **Análisis** de los aspectos que se van a cubrir durante la misma.
- **Implementación** de los requisitos fijados en el análisis.
    - Así como posibles **mejoras** pendientes de la etapa anterior.
- **Pruebas** para asegurar el correcto funcionamiento de la implementación realizada.
- **Evaluación** de los objetivos conseguidos respecto a los esperados.

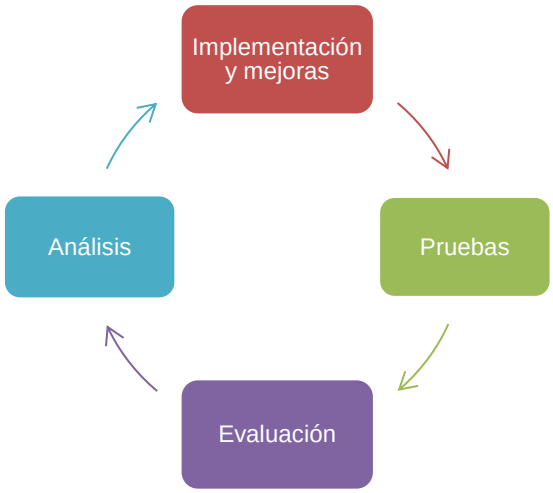


Figura 0.3.1 Fases del modelo iterativo e incremental

Este proceso ayuda al orden del desarrollo y al análisis de los fallos que han provocado el no alcanzar los objetivos deseados, brindando de mayor calidad tanto al proceso de trabajo como al producto final. También, se obtiene una gran

flexibilidad frente a los cambios debido a la retroalimentación constante cada vez que finaliza una iteración.

0.4 TECNOLOGÍA EMPLEADA

0.4.1 EVALUACIÓN DE DIVERSAS ALTERNATIVAS

El primer punto a analizar son las bases de datos. La base de datos seleccionada debe ser acorde con el tipo de problema para poder aprovechar al máximo las ventajas de esta. Las bases de datos se pueden categorizar como relacionales y no relacionales:

- En las **relacionales**, el aspecto a destacar es que su información se almacena en tablas con una estructura establecida, siguiendo un modelo relacional como su nombre indica. Se utiliza el lenguaje de programación SQL (*Structured Query Language*) para diseñar, administrar y recuperar la información de la base de datos. Algunos ejemplos son Microsoft SQL Server, Oracle, MariaDB, MySQL… [16].
- En las **no relacionales**, a diferencia de las anteriores, no se trabaja con estructuras definidas de filas y columnas. La información es almacenada en documentos, y es muy útil cuando no se tiene clara la estructura de los datos que se van a almacenar. Algunos ejemplos son MongoDB, Cassandra, Redis… [17].

Respecto a una tecnología que produzca el paso de la información de la API hacia la base de datos del sistema, se encuentra **Kafka**, aunque una alternativa podría ser Amazon **Kinesis**. Estas tecnologías permiten el flujo de mensajes y facilitan la captura y procesamiento de los datos a gran escala. Como diferencias principales, Kafka es considerablemente más complejo de configurar debido a que requiere su propio clúster, nodos, replicaciones y particiones. En cambio, Kinesis, se propone como una solución para tener lista para producción en un par de horas [4] [18].

Puesto que se va a emplear Kafka por razones expuestas en el siguiente apartado, hay que tener en cuenta los lenguajes de programación para el

desarrollo de sus microservicios. Se valora la opción de implementar tanto el Producer como el Consumer en **Java**, o uno de ellos en Java y otro en **Python**.

Con relación al desarollo de software de la aplicación web, existen diferentes pilas de desarrollo que incluyen todas las tecnologías para las diferentes fases de implementación. Algunas de las principales son:

- **MEAN** (MongoDB, Express.js, Angular.js, Node.js) [19].
- **MERN** (MongoDB, Express.js, React, Node.js) [20].
- **Django** (Django, Python y MySQL) [21].

Por último, se debe escoger una API que nos proporcione los datos para llenar el sistema de almacenamiento y sobre los que implementar la página web. Como repositorio de API's, se plantean los siguientes:

- **Kaggle.** En este Hub existen API's de recetas como *Food.com*, *recipes*, *Food Ingredients and Recipes Dataset with Images*, etc [22].
- **Rapidapi.** Incluye API's como *Recipe - Food – Nutrition*, *Yummly*, *Tasty*, etc [23].

Ambos son muy similares y ofrecen una gran cantidad de API's, por lo que la elección de uno u otro dependerá de dónde se encuentra la API final seleccionada.

0.4.2 ALTERNATIVAS ESCOGIDAS

Para el desarrollo software del proyecto, se ha escogido una base de datos **no relacional**. La principal razón es la comodidad a la hora de almacenar las recetas culinarias, las que, tras un primer análisis, se ha observado que no siguen la misma estructura. Además, gracias a que soportan grandes volúmenes de información, permiten hacer la aplicación fácilmente escalable.

- **MERN**

Teniendo en cuenta que la base de datos a emplear es no relacional se descarta Django. La solución escogida es una pila de desarrollo MERN [20]. Tras un primer contacto con MERN y con MEAN, se observó que la curva de aprendizaje de Angular era considerablemente más elevada que la de React, pudiendo obtener unos resultados similares. Este marco de desarrollo permite crear páginas web rápidas y escalables. El motivo de la elección de una pila de

desarrollo es que cubre todas las tecnologías necesarias para completar todas las capas de una aplicación web, facilitando su creación. Además, se simplifica el despliegue al contar con el servidor web integrado, se facilita el manejo de grandes volúmenes de datos gracias a la flexibilidad de MongoDB y el manejo de archivos JSON, y se obtienen aplicaciones escalables y optimizadas para su posible despliegue en la nube.

Las tecnologías que conforman este framework son:

- **MongoDB**: como se mencionó anteriormente, la base de datos empleada es no relacional. Está orientada a documentos y es de esquema libre, por lo que, gracias a ella, se obtienen ventajas como la flexibilidad a la hora de definir los esquemas de datos, la simplicidad de su uso, el alto rendimiento y la escalabilidad.
- **Express.js**: framework [6] de Node.js que facilita la organización de la aplicación web. El motivo por el que se considera interesante es por las tareas que realiza, como la asignación de rutas y el manejo de solicitudes.
- **React.js**: biblioteca de JavaScript basada en componentes que permite desarrollar el front-end de la aplicación, facilitando su diseño. En este caso, su principal ventaja es su leve curva de aprendizaje, lo que facilita la familiarización con la tecnología. También es importante el grado de personalización que ofrece. También se empleará MaterialUI como biblioteca de componentes de React, para aprovechar componentes ya implementados y diseñados.
- **Node.js**: este entorno de ejecución está destinado al lado del servidor. Escogido por su diseño para crear aplicaciones escalables, puede gestionar numerosas conexiones simultáneas y asíncronas.

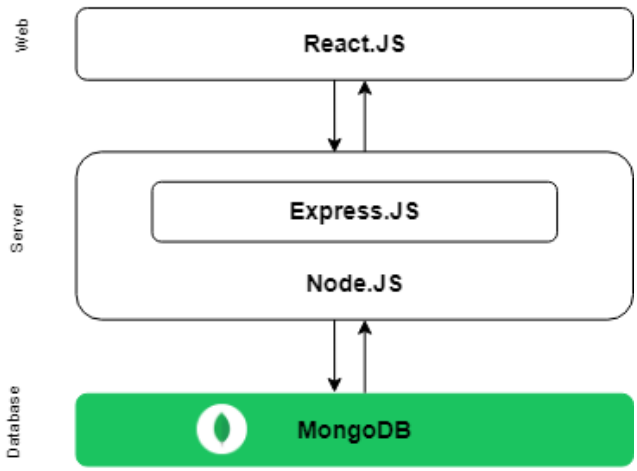


Figura 0.4.2.1 Esquema tecnologías de MERN stack

- **KAFKA**

Se ha empleado la tecnología Apache Kafka como medio de transmisión de datos con baja latencia. La finalidad de esta plataforma es administrar el paso de mensajes en tiempo real. En este proyecto, a petición de la empresa tutora, será empleada para recopilar los datos de una API con un Producer, transmitirlos a través de un topic de recetas en Kafka y leerlos con un Consumer.

Para su funcionamiento, se han desarrollado un Kafka Producer que consiste en un proyecto Maven de **Java** y un Kafka Consumer desarrollado en **Python**. **Maven** facilita la gestión del ciclo de vida del proyecto, así como las dependencias [24]. La razón de emplear dos lenguajes diferentes sigue el propósito de fomentar tanto conocimientos en ambos como hacer distinción de los dos microservicios. La independencia de estos dos servicios permite que la forma de implementar cada uno no influya en ningún aspecto del otro.

Kafka también precisa de Apache **ZooKeeper** para funcionar de manera síncrona. Es responsable de realizar un seguimiento del estado de los elementos que intervienen en Kafka (brokers del clúster, topics, particiones…) [3].

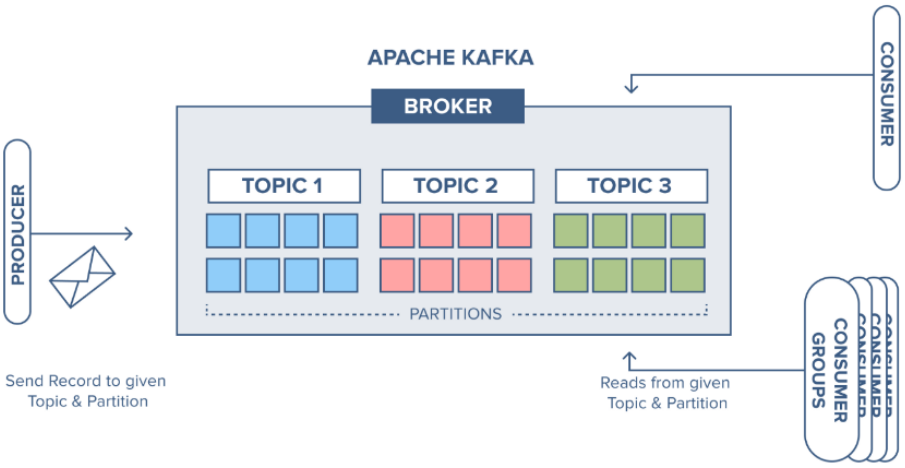


Figura 0.4.2.2 Esquema Apache Kafka [25]

- ***TASTY* DE RAPIDAPI**

Se ha escogido RapidAPI como repositorio debido a que incluye la API *Tasty*. Realizando peticiones de prueba a varias de las API's mencionadas en el apartado anterior, se ha decidido que es la que más datos interesantes ofrece. Entre ellos imágenes, categorías de recetas, país de origen de la receta, tiempo de preparación, etiquetas, etc.

Esta API ofrece diferentes planes de pago en función de las peticiones mensuales permitidas. El proyecto se realizará con el plan gratuito que incluye 500 peticiones al mes, pudiendo obtener por cada petición 40 documentos (entendiendo como documento una receta o una compilación de recetas) [26].

- **DOCKER**

Tanto la base de datos MongoDB como los servicios Kafka y ZooKeeper estarán alojados en su contenedor Docker correspondiente, formando un Docker Compose [5]. Esto facilita su despliegue y configuración desde un solo fichero "docker-compose.yml". La razón de esta elección es la facilidad de levantamiento de sistemas con diferentes tecnologías y servicios, así como la optimización de los recursos que supone frente a otras alternativas como las máquinas virtuales. Otro objetivo es fomentar los conocimientos sobre los contenedores de software.

- **HERRAMIENTAS ADICIONALES**

Como herramientas adicionales, se ha empleado:

- **Docker Desktop**: para el manejo de los contenedores que contendrán las imágenes de MongoDB, Apache Kafka y Apache Zookeper.
- **Autodesk SketchBook**: para el diseño del logo de la aplicación web.
- **Robo 3T** y **DataGrip**: para establecer una conexión con la base de datos y visualizar las colecciones y documentos almacenados en el MongoDB.
- **VisualStudio, IntelliJ IDEA y PyCharm:** para el desarrollo del código software en función al lenguaje de programación.
- **PostMan**: para realizar pruebas del funcionamiento de la API desarrollada en el back-end.
- **GitHub**: como sistema de control de versiones.
- **WhatRuns**: como extensión de Chrome que indica las tecnologías empleadas por una página web concreta.
- **App.diagrams.net** y **GoodNotes:** para la elaboración de esquemas y diagramas.
- **Microsoft Word**: para la documentación y organización del proyecto.
- **Microsoft Teams** y **Google Meet**: para la realización de las reuniones de seguimiento con los tutores correspondientes.

ESTRUCTURA DEL TRABAJO

La memoria del proyecto se estructura en cuatro apartados, los cuales tratan el proceso de desarrollo de la aplicación:

1) Estudio del problema: se describe el problema que se va a tratar, y se analizan las tecnologías, plataformas, herramientas o trabajos previos a este proyecto.
2) Gestión de proyecto software: se analiza la definición del alcance del proyecto, el plan de trabajo, la gestión de recursos, de riesgos, y la legislación y normativa vigente a tener en cuenta.
3) Solución: se trata de la parte central del proyecto. Se entra en detalle en la descripción del tipo de solución escogida, en el proceso de desarrollo, describiendo los requisitos, la fase de diseño, de implementación, y de pruebas del sistema, y en el producto de desarrollo, mostrando la aplicación implementada y explicando su funcionalidad. También se describe la forma de desplegarla.

4) Evaluación: se analiza el nivel de cumplimiento de los requisitos iniciales validando o no el producto, y se entra en detalle en la forma de evaluarlo.

1 Estudio del problema

En este apartado se presenta el contexto de realización del trabajo y se hace un análisis del estado del arte en problemas relacionados, así como se define el problema en función de la información obtenida de los puntos anteriores.

1.1 EL CONTEXTO DEL PROBLEMA

La sociedad se ha visto altamente afectada por el desarrollo tecnológico de los últimos años. Se podría decir que la mayor parte de la población de países desarrollados se ha adaptado correctamente a estos cambios. Entre las personas adaptadas a la tecnología, lo más común es disponer de un teléfono móvil inteligente, una tablet o un ordenador, y emplearlos para prácticamente cualquier aspecto de sus vidas, incluso el culinario. Por esta parte, los usuarios de Internet demandan aplicaciones en las que poder consultar y guardar recetas culinarias de manera rápida y efectiva.

Por otra parte, para los desarrolladores de front-end es interesante disponer de una arquitectura ya implementada a la que poder realizar peticiones de los datos, preocupándose solo de la tarea que les corresponde, que es el diseño.

1.2 EL ESTADO DEL ARTE

Cuando se trata de aplicaciones culinarias, las más comunes son las aplicaciones móviles y las aplicaciones web. Hay una cantidad inmensa de ellas, y hay que tener en cuenta que la mayoría de los usuarios, antes de descargarse una aplicación o recurrir siempre a la misma página web, suele hacer una búsqueda de la receta deseada en el navegador y hacer click en alguna de las primeras sugerencias que le salen. Es por esto por lo que se consideran como las principales alternativas competidoras a las páginas web con mejor posicionamiento en la lista de resultados de los navegadores (SEO [27]). En los

siguientes apartados se evalúan las ventajas y desventajas de algunas de las mismas.

1.2.1 ALTERNATIVAS SIMILARES

Se procede a hacer un análisis de las ventajas y desventajas de algunas de las páginas web principales que aparecen en el navegador Google Chrome al realizar búsquedas de recetas de comida. También, se mencionarán las tecnologías empleadas para su desarrollo detectadas por la herramienta "*whatruns*".

1.2.1.1 De Rechupete

- Ventajas: hay gran variedad de colecciones y categorías de recetas, además de artículos sobre consejos o recomendaciones.
- Desventajas: hay mucha información en la pantalla principal. Si te vas desplazando hacia el final de la página, hay numerosos artículos y una cantidad demasiado grande de información que no son recetas. Esta información se repite en el menú superior.
- Tecnologías: el lenguaje empleado es PHP, y el servidor web está alojado con Nginx. La interfaz gráfica está realizada con WordPress, con una plantilla personalizada.

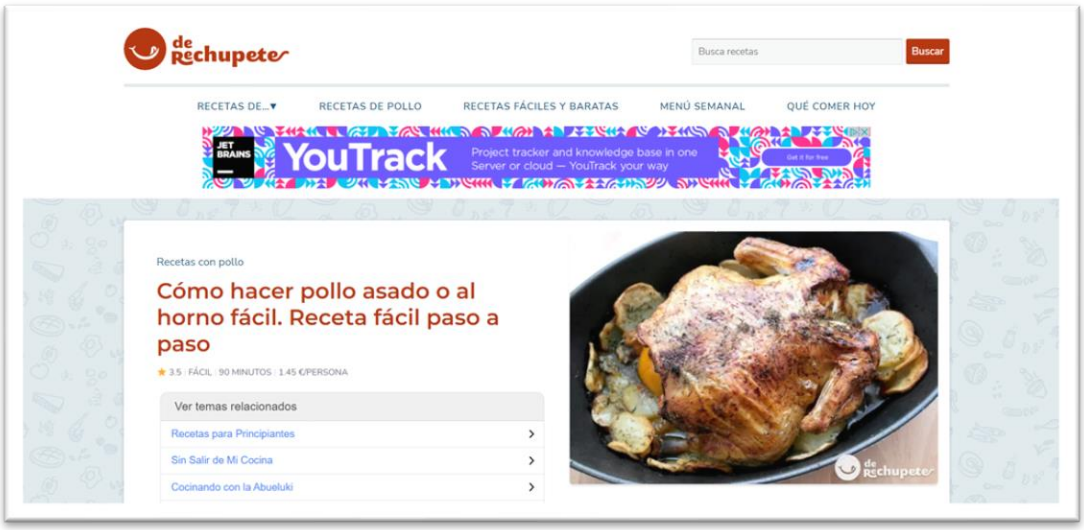


Figura 1.1 Página web *recetasderechupete.com*

1.2.1.2 Cocina casera y fácil

- Ventajas: menú lateral izquierdo con sugerencias de recetas. La aplicación es colorida y agradable a la vista.
- Desventajas: demasiados anuncios que sobrecargan la interfaz y dificultan el enfocarse en la información realmente útil. El menú con las categorías se encuentra por medio de la página, por lo que no se encuentra fácilmente. En el *footer* hay más categorías con muchos enlaces, en vez de la información necesaria de la empresa. Al adaptarla a un móvil hay componentes que se sobresalen de la pantalla.
- Tecnologías: la página web ha sido programada en PHP, y el servidor web está alojado con Nginx. La interfaz gráfica está realizada con WordPress.

Figura 1.2 Página web *cocinacaserayfacil.net*

1.2.1.3 COCINAFÁCIL (lecturas.com)

- Ventajas: buena estructuración de la información y diseño agradable y sencillo. Permite almacenar recetas al iniciar sesión.
- Desventajas: el header ocupa siempre gran parte de la página, reduciendo la visualización del contenido. No es completamente responsive.
- Tecnologías: se ha empleado HTML5, JavaScript y Bootstrap, y el servidor web está alojado con CloudFlare.

Figura 1.3 Página web *lecturas.com*

1.2.2 CONCLUSIONES

Observando las plataformas anteriores, se llega a la conclusión de que hay muchos aspectos que se pasan por alto a la hora de desarrollar una página web.

El hecho de que la página sea responsive es un punto muy importante en este tipo de aplicaciones, ya que gran parte de los usuarios acceden a ellas desde sus dispositivos móviles. Se ha observado que la mayoría de las páginas web no cumplen este factor.

El *header* y el *footer* deberían estar diseñados para cumplir con las funciones para los que estos son empleados, en vez de sobrecargarlos de información.

El emplear WordPress es una buena alternativa cuando no se tienen conocimientos sobre programar, pero el nivel de personalización al que se puede llegar es mucho menor. El uso de tecnologías que están en auge como React permiten una aplicación completamente personalizable que se adapte a todo tipo de dispositivos.

1.3 LA DEFINICIÓN DEL PROBLEMA

En el ámbito de consultar recetas, muchas personas siguen prefiriendo hacerlo a través de libros que tienen en casa, revistas o cuadernos. Sin embargo, cada vez más población lo realiza por medios digitales. Por ello es necesario adaptar la cocina a móviles y ordenadores, para que los usuarios tengan la posibilidad de revisar recetas que le han gustado o buscar otras nuevas en cualquier momento y lugar.

Analizando las desventajas de consultar recetas por medios tradicionales, destaca la pérdida de tiempo que supone encontrar una receta que sea de su agrado, y que muchas veces el medio que se está consultando ni siquiera cuenta con la misma. Además, la cantidad de recetas que pueden encontrar por estos medios está mucho más limitada, ofreciendo la tecnología un conjunto que podría considerarse ilimitado, y reunidas en un mismo lugar. Otro punto en contra es que no es posible personalizar su búsqueda, es decir, si se quiere encontrar una receta que cumpla con un tiempo de preparación, un número de calorías y para cierto número de personas, no existe la opción de seleccionar estos filtros y obtener únicamente recetas que cumplan lo solicitado.

Cuando se consultan recetas digitalmente porque no se sabe qué cocinar, hay que tener en cuenta que las personas más mayores tienden a estar menos familiarizadas con los dispositivos tecnológicos, y es por esto por lo que se debe tratar de realizar siempre un diseño simple e intuitivo. De esta forma, el aprendizaje de su manejo es rápido, evitando sobrecarga de información y tareas complejas.

2 Gestión de proyecto software

En este apartado se detalla el planteamiento del proyecto desde un punto de vista de un entorno empresarial, entrando en detalle en su estructura y en los objetivos que se deben cumplir para el despliegue de la misma. También se tratará la planificación seguida y la gestión de los recursos y costes.

2.1 ALCANCE DEL PROYECTO

2.1.1 DEFINICIÓN DEL PROYECTO

El proyecto se puede dividir en dos subproyectos, siendo el primero un sistema de recopilación, almacenamiento y peticiones de datos y el segundo una aplicación web.

En el primer sistema, la solución implementada está formada por los microservicios Kafka y la API REST, la cual lleva a cabo toda la comunicación y peticiones a la base de datos. Los microservicios Kafka son completamente independientes, y, su función, como se ha mencionado ya en otros apartados, es el llenado de la base de datos con un total de 6000 recetas obtenidas de la API pública *Tasty*.

La segunda parte, es una aplicación web para la búsqueda y guardado de recetas culinarias. Esta parte se corresponde con el front-end, y se halla la estructura y el estilo de la página web, así como el contexto del usuario que ha iniciado sesión. Desde aquí, se generan peticiones y se crean datos que son enviados al back-end. Los usuarios pueden acceder desde cualquier dispositivo y consultar todas las recetas disponibles. En caso de no tener cuenta registrada, podrán crearse una e iniciar sesión con usuario y contraseña si desean marcar recetas como favoritas para consultarlas más tarde.

El objetivo del diseño de la aplicación web es ofrecer una interfaz simple e intuitiva para los que estén menos familiarizados con la tecnología y para los que busquen descubrir recetas de manera rápida y visual. Para ello, se evita la

sobrecarga de información y de menús redundantes, mostrando diferentes recetas y categorías en la página principal y un menú siempre visible en el *header,* desde donde el usuario podrá realizar búsquedas, cerrar sesión o acceder a sus favoritos. Para permitir una visualización rápida se muestra una foto correspondiente a cada receta.

Por otra parte, la página se adapta a los diferentes tamaños de pantallas de los dispositivos electrónicos, lo que permite acceder a ella tanto desde un ordenador, como un móvil o tablet.

El producto de la aplicación se encuentra en una primera versión, donde únicamente está disponible el idioma inglés. Ya que las recetas son solicitadas a una API ya existente que no dispone de los datos en castellano, se propone como futura mejora la adaptación del texto de estas a otros de los principales idiomas más hablados, así como la implementación de nuevas funcionalidades.

2.1.1 OBJETIVOS

Los objetivos a tratar en este punto se basan en el alcance del proyecto. Así, se pueden definir estos como la implementación de:

- Microservicio Kafka Producer que se encargue de solicitar datos a la API externa y transmitirlos a través de Kafka.
- Microservicio Kafka Consumer que se encargue de leerlos de Kafka y almacenarlos en una base de datos.
- Sistema de almacenamiento de datos que responda de manera ágil.
- Servicio API REST que maneje las peticiones dirigidas al sistema de almacenamiento.
- Aplicación web *responsive*.
- Aplicación web funcional siempre y cuando la arquitectura esté desplegada.
- Aplicación web funcional en los diferentes navegadores.

2.2 PLAN DE TRABAJO

En esta sección se presenta la planificación de las tareas, organizadas en diferentes iteraciones. Se plantea un diagrama de Gantt realizado con Excel donde se puede ver la duración de tiempo empleada para cada tarea, así como su fecha de inicio y de fin.

Antes de entrar en detalle sobre cada iteración de la implementación del proyecto, se plantean tres fases generales:

- **Inicio del proyecto:** se realizan reuniones con el cliente para estudiar el problema actual y fijar los objetivos. Se definen los requisitos de la aplicación, se establece un presupuesto y se realiza un análisis de las tecnologías.
- **Proceso iterativo e incremental:** se distribuyen las tareas en iteraciones trisemanales, y se comienza con el desarrollo del proyecto. Cada vez que pasan tres semanas y se completa una fase, se cuenta con un producto entregable funcional que se revisará con el cliente para definir posibles cambios.
- **Validación y entrega:** se realizan pruebas de validación alfa sobre todo el sistema para localizar posibles errores [14]. También, se revisan los requisitos cumplidos y se entrega el producto al cliente final.

2.2.1 IDENTIFICACIÓN DE TAREAS

Existen una serie de tareas que conforman cada una de las fases mencionadas anteriormente. A continuación, se entra en detalle sobre las mismas.

2.2.1.1 INICIO DEL PROYECTO

Se lleva a cabo una reunión con el cliente donde se tratará el problema a resolver y ambas partes compartirán pensamientos y preferencias. Aquí, se establecen los requisitos del producto y demás preferencias que tenga el cliente.

Por otra parte, se realiza un análisis del estado del arte sobre otros proyectos similares del mercado, así como una investigación y valoración de las tecnologías actuales para determinar cuáles utilizar.

2.2.1.2 PROCESO ITERATIVO E INCREMENTAL

Dentro del proceso iterativo e incremental se encuentran las siguientes iteraciones, las cuales tienen una duración establecida de tres semanas:

- **Iteración 1:** se realizan los diseños de los prototipos web y de la arquitectura, se desarrolla un archivo Docker para el despliegue de la base de datos y los sistemas Kakfa en un contenedor, se implementa el Kafka producer encargado de solicitar datos a la API y se realizan pruebas sobre las implementaciones realizadas.
- **Iteración 2:** se implementa el Kafka consumer que lea los datos de Kafka, se aplica un procesamiento a esos datos y se insertan en el sistema de almacenamiento, se desarrollan las peticiones que va a recibir el back-end y sus respuestas, y se realizan pruebas sobre lo implementado.
- **Iteración 3:** se crea el proyecto front-end y se estructura en componentes, se desarrolla la vista principal de la aplicación, se implementa la funcionalidad de la vista principal, así como los estados necesarios, y se realizan pruebas sobre lo implementado.
- **Iteración 4:** se desarrolla la página de login, la página de registro, las funcionalidades de ambos, se desarrolla el enrutamiento entre las páginas, y se realizan las pruebas sobre lo implementado.
- **Iteración 5:** se implementa un modal para ver información completa de cada receta, se desarrolla la página de favoritos del usuario, se implementa la lógica de los favoritos, y se realizan pruebas sobre lo implementado.
- **Iteración 6:** se perfecciona la lógica de la barra de búsqueda, la paginación, la lógica de seleccionar una categoría para mostrar sus recetas, y se realizan pruebas sobre lo implementado.

2.2.1.3 VALIDACIÓN Y ENTREGA

Como se mencionó anteriormente, a parte de las comprobaciones realizadas sobre los resultados de cada iteración, se llevan a cabo pruebas alfa sobre el producto terminado. En estas pruebas el cliente realiza acciones sobre la aplicación como un usuario final haría. El desarrollador está presente durante la realización de estas acciones, supervisando. De esta forma, si surge algún error

durante la ejecución de la misma podrá tomar nota de qué tiene que solucionar. La reunión con el cliente se repetiría una vez corregidos los fallos de la aplicación, entregando aquí el producto final. También se lleva a cabo un repaso de los requisitos que se habían establecido inicialmente para comprobar los que han sido cumplidos y los que no e informar debidamente al cliente.

2.2.2 ESTIMACIÓN DE TAREAS

Respecto a la estimación de las tareas llevabas a cabo durante el proyecto, hay que tener en cuenta que no todas siguen siempre la misma duración de tiempo. Influyen aspectos como la dificultad o complejidad de las tecnologías, así como el grado de conocimiento de los desarrolladores con estas.

Por esto, la tabla que se adjunta a continuación presenta una estimación del tiempo medio de cada tarea, pudiendo haber ligeros cambios. Se entiende como un día de trabajo las 8 horas que conforman una jornada laboral.

Tabla 2.1 Tiempo estimado por tipo de tarea

TIPO DE TAREA	DURACIÓN
Reuniones	1 día
Análisis tecnologías existentes	½-1 día
Instalación tecnologías	½-1 día
Análisis estado del arte	1-2 días
Especificación de requisitos	1 día
Implementación requisitos	1-10 días
Pruebas por iteración	1-2 días
Pruebas validación producto	4-6 días

2.2.3 PLANIFICACIÓN DE TAREAS

Todas las tareas mencionadas anteriormente son representadas en un diagrama de Gantt. De esta forma, se obtiene de manera visual un esquema de

la planificación seguida para llevar a cabo el proyecto, facilitando la organización del equipo de desarrollo.

La fecha de inicio del proyecto es el lunes 10 de enero de 2022, y, por tanto, la de inicio de la primera fase. La sigue el proceso iterativo y una fase final de validación y entrega, lo que produce un total de 102 días laborales. Las fases y tareas son dependientes unas de otras, por lo que deben esperar a que finalice la anterior para poder comenzar.

TAREA	DURACIÓN	INICIO	FIN
Plan inicial	**6**	10-1-22	17-1-22
Proceso iterativo e incremental	**90**	18-1-22	23-5-22
Validación y entrega	**6**	24-5-22	31-5-22

Figura 2.1 Planificación fases del proyecto

2.2.3.1 PLAN INICIAL

Para el plan inicial del proyecto se ha establecido una duración de 6 días laborales. Con fecha de inicio el 10 de enero de 2022 y fecha de fin el 17 de enero de 2022, se han organizado las subtareas de la siguiente manera:

Aplicación web de gestión de recetas culinarias
María Álvarez Castelo

Inicio del proyecto: lu, 10/1/2022
Semana para mostrar: 1

TAREA	DURACIÓN	INICIO	FIN
Plan inicial	**6**	10-1-22	17-1-22
Reunión con el cliente y especificación requisitos	2	10-1-22	11-1-22
Estudio del problema	2	12-1-22	13-1-22
Análisis del estado del arte	1	14-1-22	14-1-22
Investigación de las tecnologías y su instalación	1	15-1-22	17-1-22

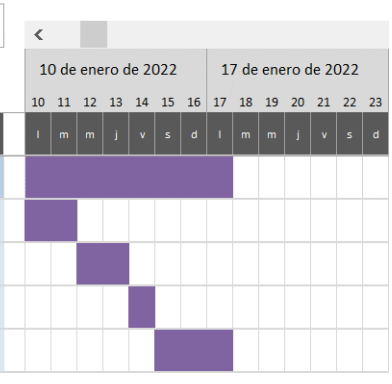


Figura 2.2 Diagrama de Gantt Plan Inicial

2.2.3.2 PROCESO ITERATIVO E INCREMENTAL

Respecto al proceso iterativo e incremental, el producto ha resultado en un total de 6 iteraciones, en las que el producto final iba adquiriendo funcionalidad. Con fecha de inicio el 18 de enero de 2022 y fecha de fin el 23 de mayo de 2022, tiene una duración de 90 días laborales.

TAREA	DURACIÓN	INICIO	FIN
Plan inicial	**6**	10-1-22	17-1-22
Proceso iterativo e incremental	**90**	18-1-22	23-5-22
Iteración 1	**15**	18-1-22	7-2-22
Iteración 2	**15**	8-2-22	28-2-22
Iteración 3	**15**	1-3-22	21-3-22
Iteración 4	**15**	22-3-22	11-4-22
Iteración 5	**15**	12-4-22	2-5-22
Iteración 6	**15**	3-5-22	23-5-22
Validación y entrega	**6**	24-5-22	31-5-22

Figura 2.3 Iteraciones del proceso iterativo

Las subtareas de cada iteración se muestran en los siguientes apartados, así como su duración.

ITERACIÓN 1:

Aplicación web de gestión de recetas culinarias
María Álvarez Castelo

Inicio del proyecto: lu, 10/1/2022
Semana para mostrar: 2

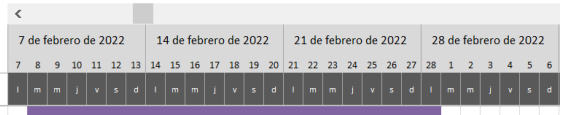
TAREA	DURACIÓN	INICIO	FIN
Iteración 1	**15**	18-1-22	7-2-22
Diseño de los prototipos web	4	18-1-22	21-1-22
Diseño de la arquitectura	4	22-1-22	27-1-22
Implementación archivo Docker compose	1	28-1-22	28-1-22
Implementación Kafka Producer	5	29-1-22	4-2-22
Pruebas	1	5-2-22	7-2-22

Figura 2.4 Diagrama de Gantt iteración 1

ITERACIÓN 2:

Aplicación web de gestión de recetas culinarias
María Álvarez Castelo

Inicio del proyecto: lu, 10/1/2022
Semana para mostrar: 5



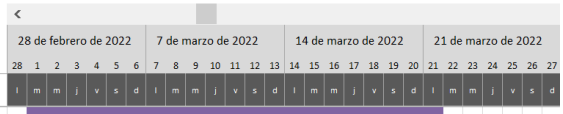
TAREA	DURACIÓN	INICIO	FIN
Iteración 2	**15**	8-2-22	28-2-22
Implementación Kakfa Consumer	5	8-2-22	14-2-22
Tratamiento e inserción de datos	3	15-2-22	17-2-22
Desarrollo peticiones back-end	6	18-2-22	25-2-22
Pruebas	1	26-2-22	28-2-22

Figura 2.5 Diagrama de Gantt iteración 2

ITERACIÓN 3:

Aplicación web de gestión de recetas culinarias
María Álvarez Castelo

Inicio del proyecto: lu, 10/1/2022
Semana para mostrar: 8



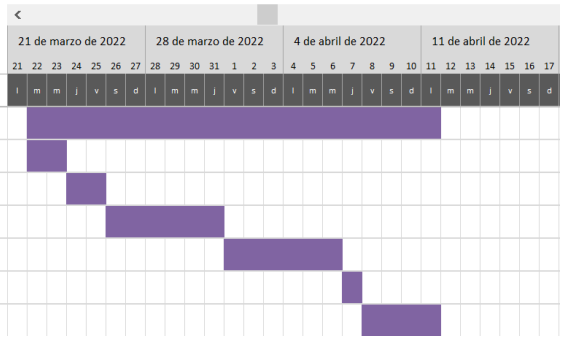
TAREA	DURACIÓN	INICIO	FIN
Iteración 3	**15**	1-3-22	21-3-22
Estructuración de componentes del proyecto front-end	2	1-3-22	2-3-22
Desarrollo estructura de la vista principal	4	3-3-22	8-3-22
Peticiones y funcionalidad en la vista principal	4	9-3-22	14-3-22
Implementación de estados necesarios	3	15-3-22	17-3-22
Pruebas	2	18-3-22	21-3-22

Figura 2.6 Diagrama de Gantt iteración 3

ITERACIÓN 4:

Aplicación web de gestión de recetas culinarias
María Álvarez Castelo

Inicio del proyecto: lu, 10/1/2022
Semana para mostrar: 11



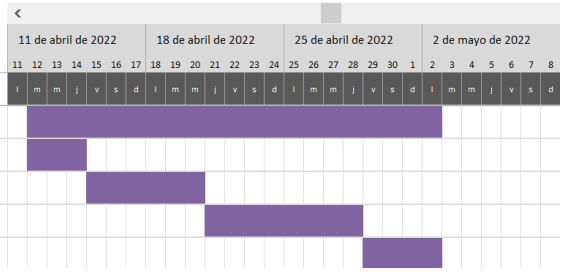
TAREA	DURACIÓN	INICIO	FIN
Iteración 4	**15**	22-3-22	11-4-22
Desarrollo página login	2	22-3-22	23-3-22
Desarrollo página registro	2	24-3-22	25-3-22
Lógica login	4	26-3-22	31-3-22
Lógica registro	4	1-4-22	6-4-22
Enrutamiento entre las páginas	1	7-4-22	7-4-22
Pruebas	2	8-4-22	11-4-22

Figura 2.7 Diagrama de Gantt iteración 4

ITERACIÓN 5:

Aplicación web de gestión de recetas culinarias
María Álvarez Castelo

Inicio del proyecto: lu, 10/1/2022
Semana para mostrar: 14



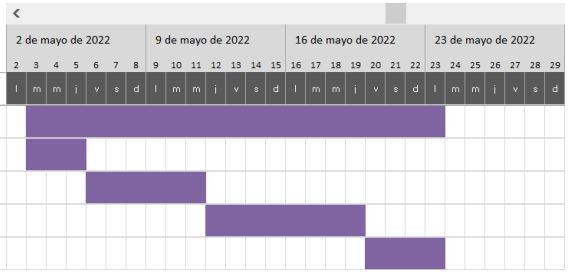
TAREA	DURACIÓN	INICIO	FIN
Iteración 5	**15**	12-4-22	2-5-22
Desarrollo modal de recetas individuales	3	12-4-22	14-4-22
Desarrollo página de favoritos	4	15-4-22	20-4-22
Lógica guardar y cargar favoritos	6	21-4-22	28-4-22
Pruebas	2	29-4-22	2-5-22

Figura 2.8 Diagrama de Gantt iteración 5

ITERACIÓN 6:

Aplicación web de gestión de recetas culinarias
María Álvarez Castelo

Inicio del proyecto: lu, 10/1/2022
Semana para mostrar: 17



TAREA	DURACIÓN	INICIO	FIN
Iteración 6	**15**	3-5-22	23-5-22
Desarrollo barra de búsqueda	3	3-5-22	5-5-22
Desarrollo paginación	4	6-5-22	11-5-22
Lógica seleccionar categoría	6	12-5-22	19-5-22
Pruebas	2	20-5-22	23-5-22

Figura 2.9 Diagrama de Gantt iteración 6

2.2.3.3 VALIDACIÓN Y ENTREGA

En cuanto a la validación y entrega final, esta comienza el 24 de mayo de 2022, y finaliza el 31 de mayo de 2022. Su duración es de 6 días, y está formada por las siguientes tres subtareas:

Aplicación web de gestión de recetas culinarias
María Álvarez Castelo

Inicio del proyecto: lu, 10/1/2022
Semana para mostrar: 20

TAREA	DURACIÓN	INICIO	FIN
Validación y entrega	**6**	24-5-22	31-5-22
Pruebas alfa y de sistema	3	24-5-22	26-5-22
Revisión de los requisitos cumplidos	2	27-5-22	30-5-22
Entrega al cliente	1	31-5-22	31-5-22

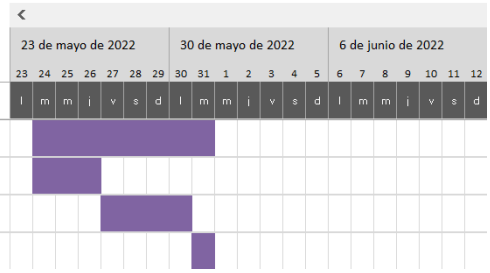


Figura 2.10 Diagrama de Gantt Validación y entrega final

2.3 GESTIÓN DE RECURSOS

En esta sección, se hace una especificación de los recursos humanos, hardware y software necesarios para el completo desarrollo del proyecto, y, acto seguido, se asignan estos recursos a sus tareas correspondientes, haciendo un análisis del coste total.

2.3.1 ESPECIFICACIÓN DE RECURSOS

A diferencia de otros tipos de proyectos, cuando se habla de recursos necesarios en una aplicación de desarrollo software están implicadas principalmente las personas que llevan a cabo la implementación de la aplicación.

A continuación, se adjunta una tabla con el coste por hora trabajada de cada tipo de trabajador requerido para el todo el proceso software. Los datos han sido obtenidos de la página web *Indeed*, y se corresponden con los salarios brutos medios anuales para cada puesto en España en el año 2022 [8]

Tabla 2.2. Recursos humanos

RECURSOS HUMANOS

Recurso	Salario anual	Salario mensual	Salario por hora
Diseñador web	19.511 €	1.393,64 €	9,38 €
Desarrollador front-end	33.643 €	2.403,07 €	16,17 €
Desarrollador back-end	34.227 €	2.444,79 €	16,45 €
DBA senior	33.432 €	2.388 €	16,07 €
Desarrollador software	29.819 €	2.129,93 €	14,33 €

Se ha obtenido el salario por hora trabajada para poder realizar un cálculo del coste total del proyecto. Las fórmulas empleadas son:

$$Salario\ mensual\ =\ salario\ anual\ /\ 14 \qquad (2.1)$$

$$Salario\ por\ hora\ =\ salario\ mensual\ /\ (40\ *\ 52) \qquad (2.2)$$

En la fórmula 2.1, el salario se divide en 14 pagas: 12 meses naturales y 2 extra.

En la fórmula 2.2, 40 son las horas semanales trabajadas, y 52 las semanas que tiene un año común.

Respecto a estos sueldos, hay que tener en cuenta a mayores los impuestos de los que se hace cargo la empresa por cada trabajador. Las cifras son las siguientes:

- Contingencias comunes: 23,60%
- Formación: 0,60%
- Desempleo: 5,50%
- Accidentes de trabajo y enfermedades profesionales: 3%
- FOGASA: 0,20%

Tabla 2.3 Impuestos aplicados a los salarios brutos

IMPUESTOS A CARGO DE LA EMPRESA						
Recurso	**Contingencias**	**Desempleo**	**Riesgos laborales**	**Formación**	**Fogasa**	**Coste anual**

Diseñador web	4.604,60	1.073,105	585,33	117,06	39,02	**6.419,12**
Desarrollador front-end	7.939,75	1.850,37	1.009,29	201,86	67,29	**11.068,56**
Desarrollador back-end	8.077,57	1.882,49	1.026,81	205,36	68,45	**11.260,68**
DBA senior	7.889,95	1.838,76	1.002,96	200,59	66,86	**10.999,12**
Desarrollador software	7.037,28	1.640,05	894,57	178,91	59,64	**9.810,45**

En la tabla 2.3 se indica el coste anual que le supone a la empresa los impuestos sobre los trabajadores. Para averiguar el coste real, se calcula el coste por hora de cada uno de estos utilizando las fórmulas 2.1 y 2.2. Así, los costes de impuestos por hora trabajada son los de la taba 2.4.

Tabla 2.4 Coste de recursos humanos con impuestos

IMPUESTOS RECURSOS HUMANOS			
Recurso	**Coste anual**	**Coste mensual**	**Coste por hora**
Diseñador web	6.419,115	458,508214	0,22
Desarrollador front-end	11.068,56	790,611429	0,38
Desarrollador back-end	11.260,68	804,334286	0,39
DBA senior	10.999,12	785,651429	0,38
Desarrollador software	9.810,45	700,746429	0,34

También, hay que tener en cuenta los costes de los dispositivos software. En la siguiente tabla se muestra el coste por unidad de cada recurso, las unidades necesarias y su importe, así como la suma de unidades y de los importes totales.

Tabla 2.5. Recursos software

RECURSOS SOFTWARE			
Recurso	**Coste unitario**	**Unidades**	**Coste total**
Microsoft 365	79 €	1	79 €
Windows 10 PRO	259 €	1	259 €
TOTAL		**2**	**338 €**

Por otra parte, los costes de hardware, agua, luz, calefacción y derivados están incluidos en los costes indirectos.

2.3.2 ASIGNACIÓN DE RECURSOS

En este apartado se va a realizar una asignación de los recursos anteriores para este proyecto, a fin de estimar un presupuesto final.

Respecto a los recursos humanos, realizarán jornadas laborales de 8 horas diarias. Las horas totales asignadas a cada uno se reflejan en la siguiente tabla, mostrando el coste total de cada recurso:

Tabla 2.6 Coste de recursos humanos

COSTE DE RECURSOS HUMANOS				
Recurso	**Horas de trabajo**	**Coste sin impuestos**	**Coste de impuestos**	**Coste total**
Diseñador web	80	750,4 €	17,63 €	768,03 €
Desarrollador front-end	240	3.880,8 €	91,22 €	3.972,02 €
Desarrollador back-end	280	4.606 €	108,28 €	4.714,28 €
DBA senior	56	899,92 €	21,15 €	921,07 €
Desarrollador software	88	1.261,04 €	29,65 €	1.290,69 €

TOTAL	**744**	**11.398,16 €**	**267,94 €**	**11.666,09 €**

Las horas totales trabajadas ascienden a 744, lo que se traduce en 93 días laborales de despliegue de la arquitectura, diseño, desarrollo y pruebas. El coste de estas horas trabajadas para la empresa suma un total de 11.666,09 €.

Respecto a los recursos software, se utilizarán todos los especificados en el apartado anterior. Puesto que el plan de la API *Tasty* que se está utilizando es el gratuito, los costes ascienden 338 €.

También, se tendrá en cuenta el beneficio industrial que será del 11%, los costes indirectos y el IVA.

2.4 PRESUPUESTO

Con relación al presupuesto disponible para la realización del proyecto, hay que tener en cuenta los costes previamente analizados.

Tabla 2.7. Costes totales del proyecto

COSTES TOTALES	
Tipo de recurso	**Coste**
Humanos	11.666,09 €
Software	338 €
Beneficio industrial	1.283,27 €
Costes indirectos	1.800,61 €
IVA	3.168,47 €
TOTAL	**18.256,44 €**

Por tanto, el total del desarrollo del producto asciende a los 18.256,44 €. Para contar con un margen ante gastos imprevistos o posibles retrasos que supongan costes adicionales, el presupuesto se fija en un total de 20.000 €.

2.5 LEGISLACIÓN Y NORMATIVA

El proyecto actual está situado bajo el marco de las leyes incluidas a continuación.

- **Ley de propiedad intelectual** (https://www.boe.es/eli/es/rdlg/1996/04/12/1)

  El Real Decreto legislativo 1/1996, del 12 de abril, establece en el artículo 1 que la propiedad intelectual de una obra ya sea literaria, artística o científica corresponde al autor por el solo hecho de su creación. En el artículo 10.1 sección "i", se especifica que como obra están incluidos los programas informáticos, así como cualquier diseño relacionado con una pintura o boceto.

  Por ello, el uso tanto de la arquitectura software como del diseño y logo de la página web quedan exclusivamente a disposición y explotación de su autor.

  Como se especifica más adelante, la arquitectura software será en futuras versiones de uso público, dejándose de aplicar por ello esta ley en este sector.

- **Ley Orgánica de protección de datos personales y garantía de los derechos digitales** (https://www.boe.es/boe/dias/2018/12/06/pdfs/BOE-A-2018-16673.pdf)

  En cumplimiento de la Ley Orgánica 3/2018, del 5 de diciembre, la protección de las personas físicas en relación con el tratamiento de datos personales es un derecho fundamental protegido por el artículo 18.4 de la Constitución española.

  Los datos recopilados en este proyecto sobre los usuarios cumplen los principios ARCO (Acceso, Rectificación, Cancelación, Oposición). El usuario desde las interfaces de la aplicación dispondrá de un correo electrónico de contacto, al que se pueden realizar las peticiones o quejas oportunas.

3 Solución

En esta sección se exponen las decisiones adoptadas para llevar a cabo la solución del problema planteado en el capítulo 1. Se entrará en detalle en las fases del proceso de desarrollo de análisis, diseño, implementación y pruebas, así como también se especificarán los requisitos funcionales y no funcionales.

3.1 DESCRIPCIÓN DE LA SOLUCIÓN

La solución adoptada resulta en un sistema completo que ha sido dividido en dos subsistemas y sus servicios correspondientes.

- Sistemas de tratamiento y obtención de datos:
  1. *Docker-compose.yml* para despliegue de Kafka, Zookeeper y MongoDB.
  2. Peticiones a la API Tasty para la obtención de los datos.
  3. Transmisión de los datos a través de Apache Kafka.
  4. Tratamiento de los campos de los datos.
  5. Almacenamiento de los datos tratados en colecciones de MongoDB.
  6. API REST a través de la cual interactuar con el sistema de almacenamiento MongoDB.
- Aplicación web:
  1. Aplicación web que permite la visualización y la interacción con estos datos.

El primer subsistema está planteado de forma que en un futuro se pueda lanzar como una solución independiente. Aquellos que quieran disponer de una base de datos MongoDB que incluya una colección de recetas culinarias y otra de categorías, cuentan con un servicio ya implementado que realiza todo el proceso de solicitud de los datos externos, tratamiento, almacenamiento y peticiones de los datos internos. Los usuarios interesados pueden diseñar y programar una aplicación tanto móvil, web o de escritorio de recetas culinarias en las tecnologías que deseen, despreocupándose de la obtención de los datos. Una vez se ejecuten los sistemas y se disponga de las recetas en la base de datos,

solo deberán realizar las peticiones a las consultas implementadas en el back-end.

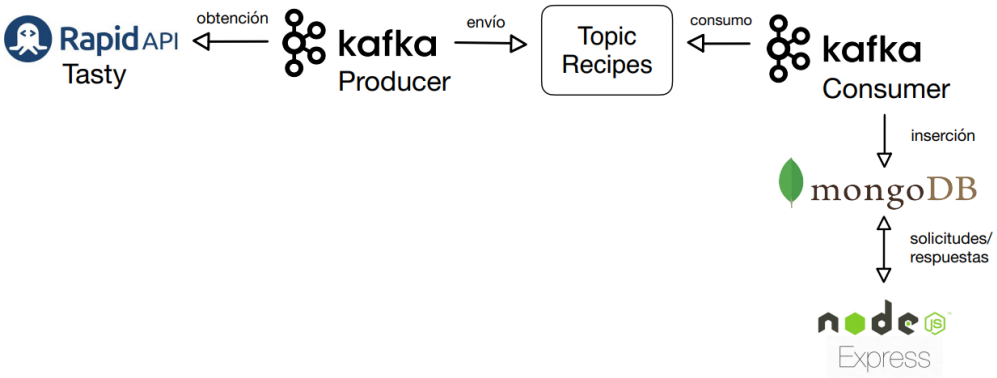


Figura 3.1. Sistema de datos

Mientras tanto, en este proyecto, esta solución es empleada para la aplicación web Zucca. Los interesados son usuarios de Internet que buscan descubrir recetas de cocina. Se plantea esta aplicación web como una solución obtenida a partir del sistema anterior, de forma que con los mismos datos es posible generar gran variedad de aplicaciones con diferentes diseños y funcionalidades. El diseño es *responsive* para permitir acceder desde diferentes dispositivos y que la aplicación siga siendo sencilla y funcional, aunque la mayor parte del tráfico en este contexto se realiza desde teléfonos móviles.

El funcionamiento de esta aplicación web busca la simplicidad, ofreciendo al usuario nada más entrar en la web una página principal donde se le muestran categorías y recetas. En el header, hay información fija como el icono de la aplicación para volver a la página principal y una barra de búsqueda para obtener las recetas relacionadas con esa búsqueda. La información dinámica depende de si el usuario está logueado o no. Si sí lo está, se le mostrará su nombre de usuario y su foto de perfil, en la que al hacer click podrá seleccionar en un menú si ir a su página de favoritos o cerrar sesión. Si no lo está, tiene la opción de ir a la página de iniciar sesión o a la de crearse una nueva cuenta.

Cuando el usuario actual visita su página de favoritos, puede consultar qué recetas ha guardado a lo largo del tiempo. Si vuelve a la página principal, puede marcar como favoritas nuevas recetas.

Si el usuario cierra sesión, se le dirigirá a la página de iniciar sesión. Desde aquí puede introducir de nuevo las credenciales de una cuenta existente, dirigirse a la página de crearse una nueva con un email, nombre de usuario que no exista en el sistema y contraseña o ir a la página principal.

A continuación, se adjunta un esquema donde los recuadros en tono naranja claro representan páginas de la aplicación, y los azules acciones que se pueden llevar a cabo en cada una.

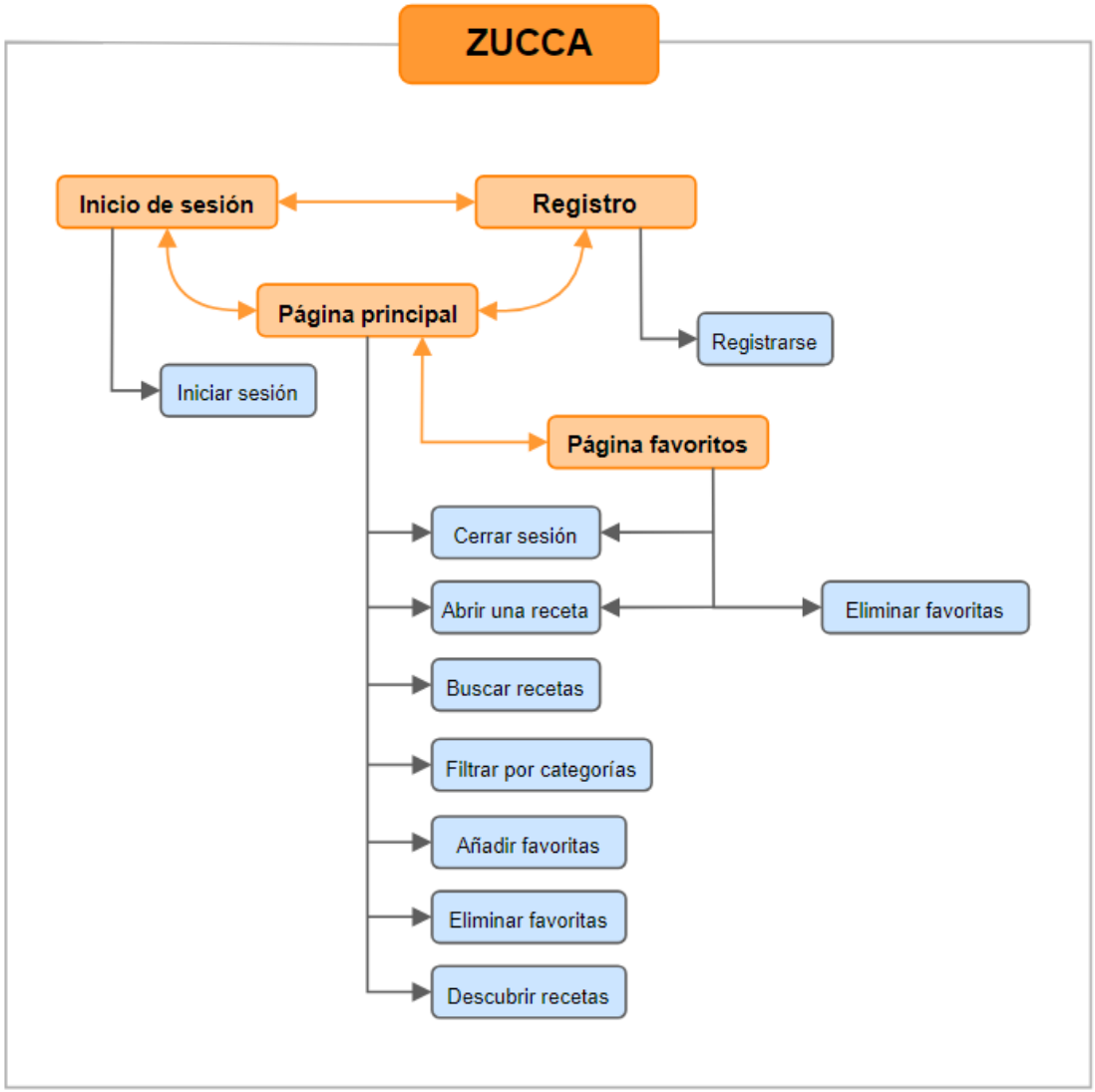


Figura 3.2. Acciones disponibles en la página web

3.2 EL PROCESO DE DESARROLLO

El proceso de desarrollo del proyecto está compuesto de cuatro fases. La primera es un análisis de los requisitos del sistema, la segunda el diseño de la aplicación, la tercera el proceso de implementación y la cuarta las pruebas del sistema.

3.2.1 ANÁLISIS DE REQUISITOS

La fase de análisis de requisitos se divide en la definición y la especificación, donde se enumerarán y se describirán los requisitos funcionales y no funcionales. Se adjuntan en formato de tabla para facilitar y agilizar su visualización.

3.2.1.1 DEFINICIÓN DE REQUISITOS

Los requisitos del sistema fueron establecidos en la reunión inicial llevada a cabo con el cliente. Estos requisitos son divididos en no funcionales y funcionales.

Los no funcionales hacen referencia a aquellas especificaciones que debe cumplir el sistema. Están relacionados con tolerancia a fallos, rendimiento, usabilidad, seguridad…

Por otra parte, los funcionales están directamente relacionados con las funcionalidades del sistema.

Para cada uno de los no funcionales, se indica su referencia y su título, estando categorizados en requisitos de interfaz y usabilidad, tecnologías y lenguajes, rendimiento, seguridad, portabilidad, tolerancia a fallos y mantenibilidad.

Tabla 3.1. Definición de requisitos no funcionales

REQUISITOS NO FUNCIONALES
Requisitos de interfaz y usabilidad
RNF01 – Página web *responsive*
RNF02 – Diseño simple e intuitivo
RNF03 – Tonos de la aplicación en colores claros

Requisitos de tecnologías y lenguajes
RNF04 – Front-end con React
RNF05 – Back-end con Node.js y Express.js
RNF06 – Base de datos MongoDB
RNF07 – Kafka Producer en Java
RNF08 – Kafka Consumer en Python
RNF09 – Base de datos con 5000 recetas
Requisitos de rendimiento
RNF10 – Tiempo de respuesta ágil
RNF11 – Soporte a usuarios simultáneos
Requisitos de seguridad
RNF12 – Protección de datos del usuario
RNF13 – Cifrado de contraseñas
Tolerancia a fallos
RNF14 – Mensajes personalizados ante errores del sistema
Requisitos de portabilidad
RNF15 – Funcional en diversos navegadores web
RNF16 – Funcional en diversos sistemas operativos
Requisitos de mantenibilidad
RNF17 – Independencia de los componentes del sistema

A continuación, se adjuntan los requisitos funcionales con su referencia y su título correspondiente. En el próximo apartado se entrará en detalle sobre los mismos.

Tabla 3.2. Definición de requisitos funcionales

REQUISITOS FUNCIONALES
APLICACIÓN WEB
RF01 – Iniciar sesión
RF02 – Crearse una cuenta
RF03 – Ver recetas
RF04 – Buscar recetas
RF05 – Filtrar recetas por categorías
RF06 – Marcar recetas como favoritas
RF07 – Consultar sus recetas favoritas
RF08 – Eliminar recetas favoritas
RF09 – Cerrar sesión
RF10 – Volver a la página principal
RF11 – Obtener más información de cada receta
RF12 – Cambiar la página de recetas actual
DOCKER
RF13 – Desplegar contenedores con Kafka, ZooKeeper y MongoDB
SISTEMAS KAFKA
RF14 – Pedir datos a la API *tasty* con el producer
RF15 – Publicar datos en un *topic* de Kafka con el producer
RF16 – Leer datos de un topic de Kafka con el consumer
RF17 – Separar datos en recetas y categorías con el consumer
RF18 – Inserción de datos en MongoDB con el consumer
API REST
RF19 – Gestionar las peticiones recibidas

RF20 – Realizar un modelo de Usuario y Categoría

3.2.1.2 ESPECIFICACIÓN DE REQUISITOS

Para cada uno de los requisitos identificados en el apartado anterior, conviene entrar en detalle de los mismos, proporcionando, a mayores, una descripción que indique su propósito y su estado de realización.

Tabla 3.3 RNF01

REQUISITO NO FUNCIONAL	
Referencia	RNF01
Título	Página web *responsive*
Descripción	La aplicación web deberá ser *responsive* para que se ajuste correctamente a las diferentes pantallas de los dispositivos.
Tipo	Interfaz y usabilidad
Estado	Desarrollado

Tabla 3.4 RNF02

REQUISITO NO FUNCIONAL	
Referencia	RNF02
Título	Diseño simple e intuitivo
Descripción	La aplicación web deberá ser fácil de usar y visual.
Tipo	Interfaz y usabilidad
Estado	Desarrollado

Tabla 3.5 RNF03

REQUISITO NO FUNCIONAL	
Referencia	RNF03
Título	Tonos de la aplicación en colores claros
Descripción	La aplicación web deberá ser diseñada en tonos claros, manteniendo una similitud entre páginas.
Tipo	Interfaz y usabilidad
Estado	Desarrollado

Tabla 3.6 RNF04

REQUISITO NO FUNCIONAL	
Referencia	RNF04
Título	Front-end con React.js
Descripción	La interfaz de la aplicación web deberá ser diseñada con la tecnología React.js, apoyándose de MaterialUI.
Tipo	Tecnologías y lenguajes
Estado	Desarrollado

Tabla 3.7 RNF05

REQUISITO NO FUNCIONAL	
Referencia	RNF05
Título	Back-end con Node.js y Express.js
Descripción	La lógica de la aplicación web deberá ser implementada con las tecnologías Node.js y Express.js

Tipo	Tecnologías y lenguajes
Estado	Desarrollado

Tabla 3.8 RNF06

REQUISITO NO FUNCIONAL	
Referencia	RNF06
Título	Base de datos con MongoDB
Descripción	La base de datos empleada por la aplicación web deberá ser MongoDB
Tipo	Tecnologías y lenguajes
Estado	Desarrollado

Tabla 3.9 RNF07

REQUISITO NO FUNCIONAL	
Referencia	RNF07
Título	Kafka Producer en Java
Descripción	El Kafka Producer que lee los datos de la API Tasty deberá ser implementado en Java
Tipo	Tecnologías y lenguajes
Estado	Desarrollado

Tabla 3.10 RNF08

REQUISITO NO FUNCIONAL	
Referencia	RNF08
Título	Kafka Consumer en Python

Descripción	El Kafka Consumer que lee los datos del topic de Kafka, deberá ser implementado en Python
Tipo	Tecnologías y lenguajes
Estado	Desarrollado

Tabla 3.11 RNF09

REQUISITO NO FUNCIONAL	
Referencia	RNF09
Título	Base de datos con 5000 recetas
Descripción	La base de datos deberá contener al menos 5000 documentos de recetas, para asegurar la satisfacción del usuario
Tipo	Tecnologías y lenguajes
Estado	Desarrollado

Tabla 3.12 RNF10

REQUISITO NO FUNCIONAL	
Referencia	RNF10
Título	Tiempo de respuesta ágil
Descripción	El sistema, en condiciones normales, deberá asegurar unos tiempos de respuesta nunca superiores a 3 segundos para cualquier petición
Tipo	Rendimiento
Estado	Desarrollado

Tabla 3.13 RNF11

REQUISITO NO FUNCIONAL	
Referencia	RNF11
Título	Soporte a usuarios simultáneos
Descripción	El sistema, en condiciones normales, deberá soportar una carga elevada de usuarios simultáneos
Tipo	Rendimiento
Estado	Desarrollado

Tabla 3.14 RNF12

REQUISITO NO FUNCIONAL	
Referencia	RNF12
Título	Protección de datos del usuario
Descripción	La aplicación deberá respetar la ley de protección de datos
Tipo	Seguridad
Estado	Desarrollado

Tabla 3.15 RNF13

REQUISITO NO FUNCIONAL	
Referencia	RNF13
Título	Cifrado de contraseñas
Descripción	Las contraseñas de la aplicación se almacenarán cifradas en la base de datos para evitar su visualización

Tipo	Seguridad
Estado	Desarrollado

Tabla 3.16 RNF14

REQUISITO NO FUNCIONAL	
Referencia	RNF14
Título	Mensajes personalizados ante errores del sistema
Descripción	Ante diferentes errores el sistema debe mostrar al usuario mensajes claros y entendibles
Tipo	Tolerancia a fallos
Estado	Desarrollado

Tabla 3.17 RNF15

REQUISITO NO FUNCIONAL	
Referencia	RNF15
Título	Funcional en diversos navegadores web
Descripción	La aplicación deberá funcionar correctamente en los diferentes navegadores web
Tipo	Portabilidad
Estado	Desarrollado

Tabla 3.18 RNF16

REQUISITO NO FUNCIONAL	
Referencia	RNF16

Título	Funcional en diversos sistemas operativos
Descripción	La aplicación deberá funcionar correctamente en los diferentes sistemas operativos
Tipo	Portabilidad
Estado	Desarrollado

Tabla 3.19 RNF17

REQUISITO NO FUNCIONAL	
Referencia	RNF17
Título	Independencia de los componentes
Descripción	La aplicación deberá disponer de sus diferentes componentes por separado, para mejorar su mantenibilidad en el futuro
Tipo	Mantenibilidad
Estado	Desarrollado

Para la especificación de los requisitos funcionales, se indica el subsistema (vista de la página web) en la que debe estar implementado, así como posibles dependencias con otros requisitos y su estado. No se incluye el tipo de requisito, ya que son todos de funcionamiento de la aplicación.

Tabla 3.20 RF01

REQUISITO FUNCIONAL	
Referencia	RF01
Título	Iniciar sesión

Descripción	El usuario podrá iniciar sesión en la aplicación con su nombre de usuario y contraseña
Subsistema	Login
Dependencia	-
Estado	Desarrollado

Tabla 3.21 RF02

REQUISITO FUNCIONAL	
Referencia	RF02
Título	Crearse una cuenta
Descripción	El usuario podrá crearse una cuenta en la aplicación con su nombre de usuario, email y contraseña
Subsistema	Registro
Dependencia	-
Estado	Desarrollado

Tabla 3.22 RF03

REQUISITO FUNCIONAL	
Referencia	RF03
Título	Ver recetas
Descripción	En la página principal, el usuario podrá visualizar recetas generadas aleatoriamente cada vez que entre a la aplicación
Subsistema	Página principal

Dependencia	-
Estado	Desarrollado

Tabla 3.23 RF04

REQUISITO FUNCIONAL	
Referencia	RF04
Título	Buscar recetas
Descripción	En el *header* de la página principal, el usuario podrá realizar búsquedas de recetas por texto coincidente
Subsistema	Página principal
Dependencia	-
Estado	Desarrollado

Tabla 3.24 RF05

REQUISITO FUNCIONAL	
Referencia	RF05
Título	Filtrar recetas por categorías
Descripción	En la página principal, el usuario podrá seleccionar una de las sugerencias de categorías generadas aleatoriamente y se le mostrarán recetas correspondientes a esa categoría
Subsistema	Página principal
Dependencia	-
Estado	Desarrollado

Tabla 3.25 RF06

REQUISITO FUNCIONAL	
Referencia	RF06
Título	Marcar recetas como favoritas
Descripción	Un usuario autenticado podrá marcar recetas como favoritas desde la página principal
Subsistema	Página principal
Dependencia	RF01
Estado	Desarrollado

Tabla 3.26 RF07

REQUISITO FUNCIONAL	
Referencia	RF07
Título	Consultar sus recetas favoritas
Descripción	Un usuario autenticado podrá visualizar sus recetas favoritas haciendo click en su foto de perfil y en “Favorites”
Subsistema	Página de favoritos
Dependencia	RF01
Estado	Desarrollado

Tabla 3.27 RF08

REQUISITO FUNCIONAL	
Referencia	RF08
Título	Eiminar recetas favoritas

Descripción	Un usuario autenticado podrá retirar de sus favoritas las recetas que desee haciendo click en el icono del corazón
Subsistema	Página principal / Página de favoritos
Dependencia	RF01
Estado	Desarrollado

Tabla 3.28 RF09

REQUISITO FUNCIONAL	
Referencia	RF09
Título	Cerrar sesión
Descripción	Un usuario autenticado podrá cerrar sesión y será redirigido a la página de login
Subsistema	Página principal / Página de favoritos
Dependencia	RF01
Estado	Desarrollado

Tabla 3.29 RF10

REQUISITO FUNCIONAL	
Referencia	RF10
Título	Volver a la página principal
Descripción	Un usuario podrá volver a la página principal desde cualquier página siempre que haga click en el logo de la aplicación situado en la parte superior de la página web

Subsistema	Página principal / Página de favoritos / Login / Registro
Dependencia	-
Estado	Desarrollado

Tabla 3.30 RF11

REQUISITO FUNCIONAL	
Referencia	RF11
Título	Obtener más información de la receta
Descripción	Un usuario podrá hacer click en cualquier receta y obtener una ventana emergente con el título, imagen, ingredientes e instrucciones de la receta
Subsistema	Página principal / Página de favoritos
Dependencia	-
Estado	Desarrollado

Tabla 3.31 RF12

REQUISITO FUNCIONAL	
Referencia	RF12
Título	Cambiar la página de recetas actual
Descripción	El usuario dispondrá de un componente a través del cual podrá cambiar la página de recetas que está visualizando
Subsistema	Página principal / Página de favoritos
Dependencia	-
Estado	Desarrollado

Tabla 3.32 RF13

REQUISITO FUNCIONAL	
Referencia	RF13
Título	Desplegar contenedores con Kafka, ZooKeeper y MongoDB
Descripción	Mediante un archivo d*ocker-compose.yml* se deberán desplegar los contenedores con las imágenes de Kafka, ZooKeeper y MongoDB
Subsistema	Sistemas Docker
Dependencia	-
Estado	Desarrollado

Tabla 3.33 RF14

REQUISITO FUNCIONAL	
Referencia	RF14
Título	Pedir datos a la API *tasty* con el producer
Descripción	El producer deberá solicitar los datos al servicio proveedor de datos *tasty* a través de peticiones HTTP
Subsistema	Sistemas Kafka
Dependencia	-
Estado	Desarrollado

Tabla 3.34 RF15

REQUISITO FUNCIONAL

Referencia	RF15
Título	Publicar datos en un *topic* de Kafka con el producer
Descripción	El producer deberá publicar los datos en el topic “recipesZucca” de Kafka
Subsistema	Sistemas Kafka
Dependencia	RF14
Estado	Desarrollado

Tabla 3.35 RF16

REQUISITO FUNCIONAL	
Referencia	RF16
Título	Leer datos de un topic de Kafka con el consumer
Descripción	El consumer deberá consumir los datos del topic “recipesZucca”
Subsistema	Sistemas Kafka
Dependencia	RF15
Estado	Desarrollado

Tabla 3.36 RF17

REQUISITO FUNCIONAL	
Referencia	RF17
Título	Separar datos en recetas y categorías con el consumer

Descripción	El consumer deberá hacer distinción de los datos que se tratan de recetas y categorías para su inserción en la base de datos
Subsistema	Sistemas Kafka
Dependencia	RF16
Estado	Desarrollado

Tabla 3.37 RF18

REQUISITO FUNCIONAL	
Referencia	RF18
Título	Inserción de datos en MongoDB con el consumer
Descripción	El consumer deberá almacenar la información en la base de datos MongoDB
Subsistema	Sistemas Kafka
Dependencia	RF17
Estado	Desarrollado

Tabla 3.38 RF19

REQUISITO FUNCIONAL	
Referencia	RF19
Título	Gestionar las peticiones recibidas
Descripción	El servicio API REST deberá disponer de peticiones para el funcionamiento de la página web
Subsistema	API REST

Dependencia	-
Estado	Desarrollado

3.2.2 DISEÑO

Para tener clara la estructura del proyecto, en los siguientes apartados se exponen las fases realizadas para el diseño del sistema. El objetivo es reflejar las decisiones que se han tomado para cumplir las especificaciones de requisitos y los objetivos tratados en puntos anteriores.

3.2.2.1 DISEÑO DE LA ARQUITECTURA

Para el desarrollo del proyecto completo se ha optado por crear un proyecto independiente para cada parte de la arquitectura, de forma que quede definida la división de tareas.

Figura 3.3 Sub-proyectos de Zucca

Es importante recordar la división ya establecida en capítulos anteriores entre el sistema de datos y la página web. Más adelante se trata como cliente al navegador que desea acceder a la página de recetas, pero la separación implementada entre el front-end y, el back-end, base de datos y sistemas Kafka permite, como se trató en el apartado Solución, que otros desarrolladores puedan omitir el front-end y utilizar estos servicios para crear sus propias aplicaciones.

Por una parte, se podría proporcionar como solución el conjunto de los microservicios Kafka junto con el componente back-end, ya que, habiendo desplegado el Docker-compose, se dispondría de una base de datos MongoDB

con documentos para los cuales las peticiones del back-end están adaptadas. Sería responsabilidad del desarrollador la manera de visualizar o tratar estos datos y el escalarlo a un entorno de producción real.

Por otra parte, puede darse el caso de que el desarrollador disponga de la información gracias a estos servicios Kafka, pero desee una reimplementación de la parte back-end, desarrollando sus propias peticiones que se adapten a su producto.

Respecto a la arquitectura del sistema, se va a establecer el foco en la distinción entre el cliente y el servidor. En este tipo de arquitecturas, el cliente es el encargado de realizar peticiones a un servidor. Es decir, es el interesado en utilizar la aplicación, y, el servidor, por otra parte, es el encargado de devolver una respuesta. En este proyecto, se trata de una arquitectura cliente/servidor de tipo aplicación web [28].

El cliente o clientes, es el navegador o navegadores que acceden a la página web y realizan peticiones de datos. Cuando este accede a la URL de la aplicación se realiza una primera solicitud de la página web, obteniendo la vista principal.

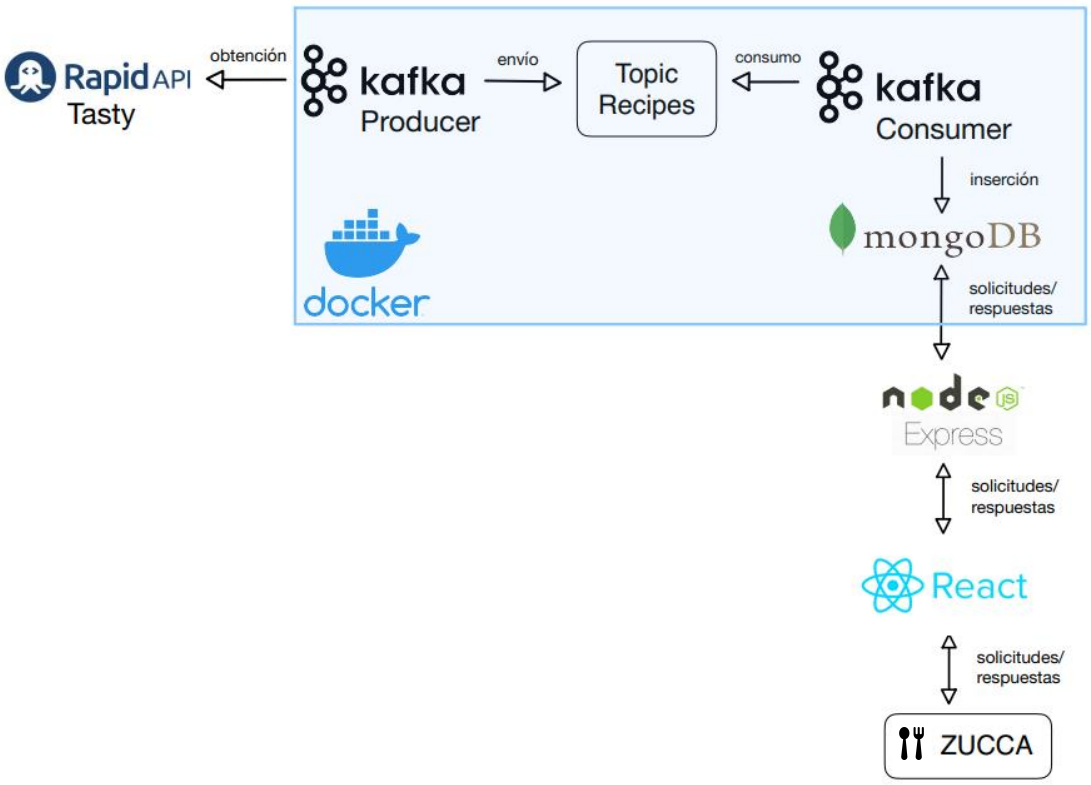


Figura 3.4 Arquitectura del proyecto

Un ejemplo real que describe el flujo de uso de la aplicación sería:

- Un cliente (navegador) accede a la aplicación web.
- La parte front-end de la aplicación tiene establecida una conexión con el servidor web (back-end) a través del puerto 5000. Mediante esta conexión se gestionan las peticiones generadas por las interacciones del usuario.
- El back-end, realiza peticiones al servidor de almacenamiento a través del puerto 27017, que es el puerto por defecto de MongoDB.

Se da por hecho que los microservicios Kafka ya han sido desplegados.

Para entrar más en detalle con el front-end, se define a este como la parte del sistema con la que el cliente interactúa de manera directa. Desarrollado con React.js y con el apoyo de MaterialUI.

En cuanto al back-end, se trata de una parte del sistema que no es accesible ni visible para el usuario. Sus conexiones son establecidas únicamente con el front-end y la base de datos. Desarrollado con Node.js y Express.js.

3.2.2.2 DISEÑO DEL FRONT-END

El sistema de front-end, emplea la biblioteca Javascript React.js para construir interfaces de usuario interactivas. Al emplear esta tecnología para el desarrollo de aplicaciones web, su uso se realiza junto con ReactDOM [29]. Estas dos tecnologías se encargan del renderizado de los componentes adecuados cuando alguno de ellos es modificado.

Para estructurar las páginas de la aplicación, se ha hecho uso de una de las características de React: los componentes. Para cada vista, y elemento que puede ser reutilizado en diferentes partes se ha creado un archivo que incluye la lógica JavaScript del componente. Dentro de este código se hallan las peticiones al back-end que soliciten o envíen la información correspondiente, los estados necesarios para gestionar cambios realizados por el usuario y la carga de todos los elementos visuales que lo forman.

El estilo aplicado a cada uno de ellos se ha apoyado en el uso de componentes de la biblioteca MaterialUI y el lenguaje CSS. MaterialUI incluye componentes ya creados y con estilo propio, facilitando en gran medida la estilización y obteniendo coherencia entre las diferentes páginas.

En la estructura interna del proyecto front-end se debe hacer distinción entre las páginas y los componentes; siendo una página una vista completa de toda una ventana del navegador, y la cual tiene su enrutamiento correspondiente, y un componente un elemento reutilizable que se halla dentro de una vista.

A parte de la posibilidad de introducir un mismo componente en varias vistas, también proporcionan encapsulación al sistema, encomendando tareas relacionadas únicamente al componente responsable. De esta forma, la aplicación tiene cuatro vistas y cuatro componentes que se muestran en el siguiente esquema. Los iconos con el símbolo de bucle indican que se realizan múltiples llamadas a ese elemento, como en el caso de generar y mostrar las recetas.

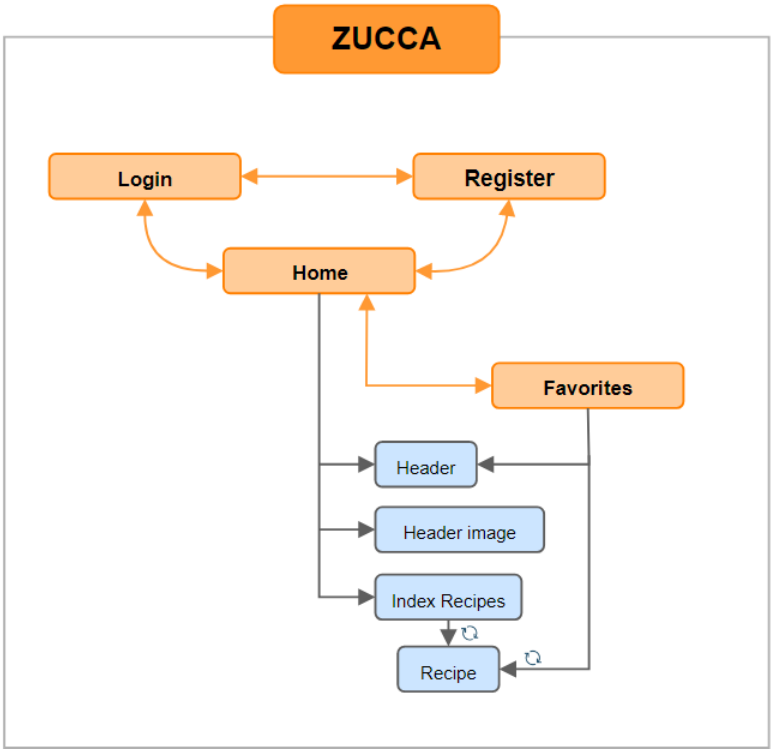


Tabla 3.39 Páginas y componentes de la aplicación web

Respecto a la estructura general de las páginas web, se establecieron en la primera iteración unos prototipos con los elementos básicos para facilitar la tarea del diseño.

La página principal y la página de favoritos cuentan con el mismo componente *header*, aunque el contenido de la parte central varíe.

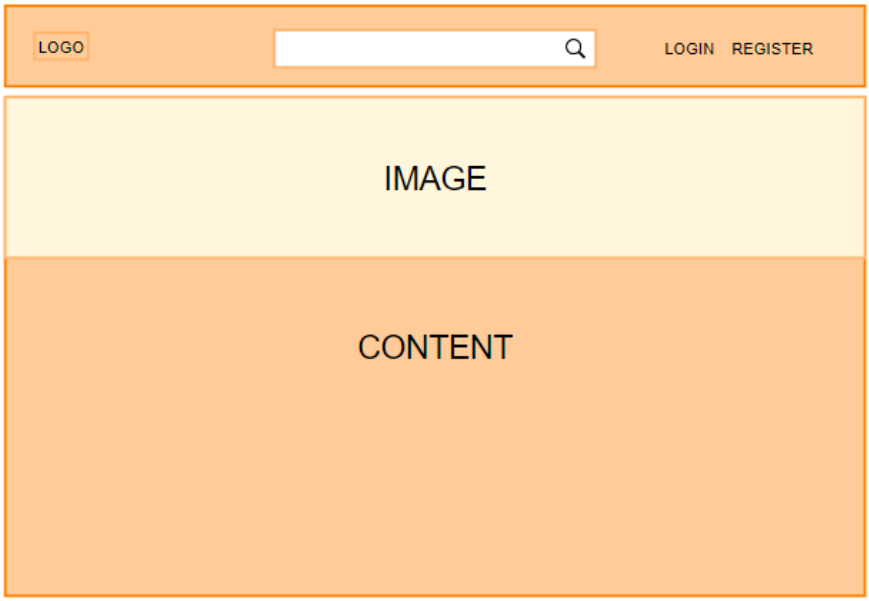


Figura 3.4 Prototipo página principal sin iniciar sesión

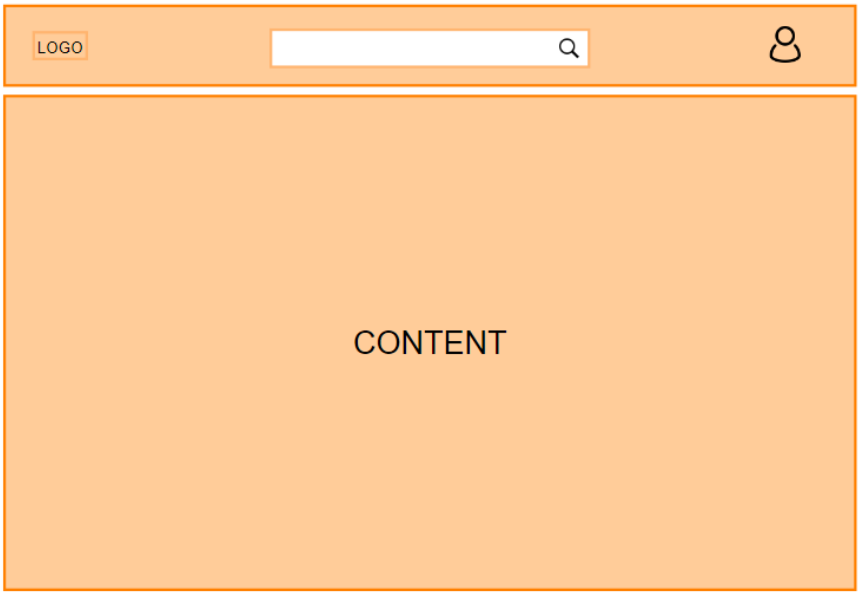


Figura 3.5 Prototipo página de favoritos

Por otra parte, la página de inicio de sesión y de registro siguen otro prototipo con un componente central en el que introducir los campos solicitados.

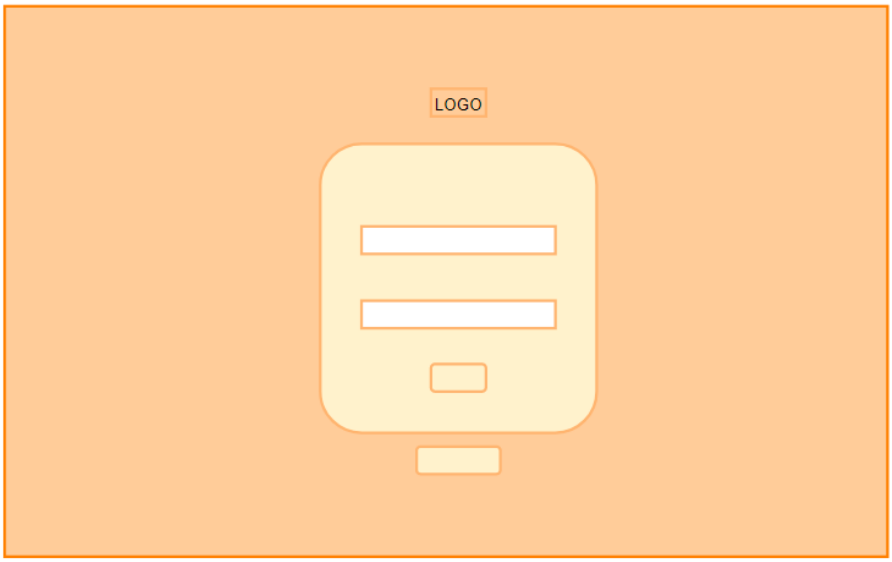


Figura 3.6 Prototipo página login y registro

Todas estas páginas web son *responsive*, adaptándose correctamente a los diferentes tamaños de los dispositivos.

Por último, otro aspecto destacable respecto al diseño de la web es la gama de colores utilizada en la aplicación. Los componentes como botones, cuadros de texto, barras de búsqueda, menú y texto, siguen la siguiente gama de colores, basada en la imagen que se encuentra en la página principal. Se observa en la figura 3.7.

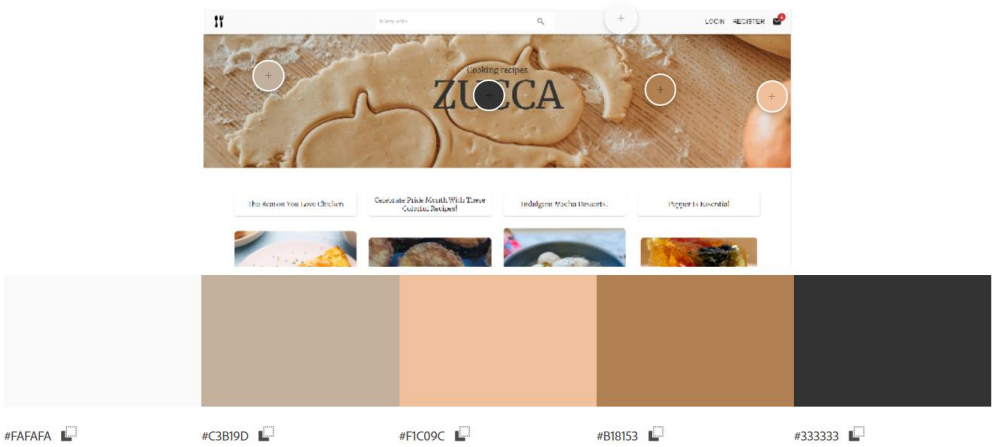


Figura 3.7 Paleta de colores página principal

En la página de registro y de login, el fondo tiene como diseño dos imágenes diferentes con colores más llamativos. Sus paletas de colores se adjuntan a continuación.

Figura 3.8 Paleta de colores seguida por la página de registro

Figura 3.9 Paleta de colores seguida por la página de login

3.2.2.3 DISEÑO DEL BACK-END

Por otra parte, está el proyecto ***back-end***, entendiendo este término, para esta estructura, como la API REST implementada en Node.js y Express.js que también emplea Mongoose. Mongoose es una librería para Node.js que permite manejar consultas para una base de datos MongoDB. En este proyecto se

aprovecha de la posibilidad de crear esquemas para los usuarios y las categorías, los cuales, a diferencia de las recetas, sí deben tener una estructura normalizada.

El back-end es el encargado de tratar las peticiones HTTP recibidas desde el front-end, y realiza la comunicación con la base de datos y pide y almacena información como recetas, categorías y usuarios. Incluye, como todos los proyectos JavaScript/Node, un *package.json* con las dependencias del proyecto, información sobre el autor y versión, entre otros [30].

Hay un archivo *config.js* con las variables que se van a necesitar de manera global, y un *index.js* que las emplea para establecer la conexión con la base de datos y levantar el servidor en el puerto deseado. También se encarga de enrutar las peticiones a su módulo correspondiente, los cuales se encuentran en *routes*. Uno se corresponde con el sistema de autenticación de usuarios (login y registro), otro con las peticiones relacionadas con recetas, y otro con un usuario, que se encarga de las peticiones relacionadas con sus favoritos y sus datos.

En la carpeta *models*, hay dos ficheros correspondientes a un usuario y una compilación (categoría). Cada uno de ellos definen un esquema de datos de su entidad correspondiente.

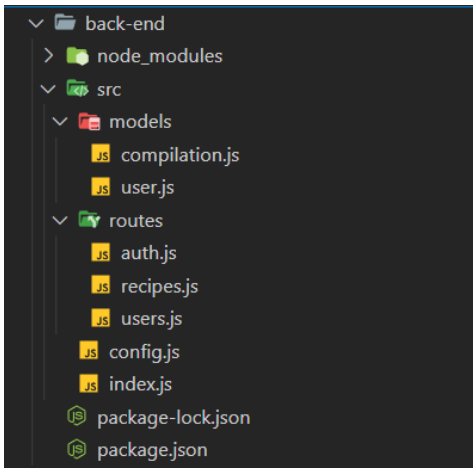


Figura 3.10 Estructura *back-end*

3.2.2.4 DISEÑO DEL DOCKER-COMPOSE

El proyecto de los sistemas docker es *docker-services,* la cual incluye el archivo que despliega el contenedor Docker que contiene la base de datos MongoDB, Kafka y ZooKeeper. Este archivo tiene la extensión "*yml*". El proyecto también incluye la carpeta de persistencia de la base de datos, cuya ruta se especifica en el fichero "yml" [31]. Se incluye un *README.md* con los comandos para ejecutar el archivo.

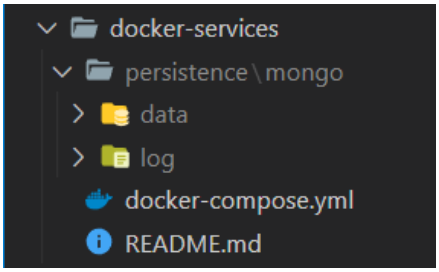


Figura 3.11 Estructura proyecto *docker-services*

3.2.2.5 DISEÑO DE LOS MICROSERVICIOS KAFKA

El último de los subproyectos de la arquitectura es ***microservices***. Está compuesto por el Kafka-consumer y el Kafka-producer, cada uno en su carpeta correspondiente puesto que uno está programado en Java y el otro en Python y se ejecutan de manera independiente.

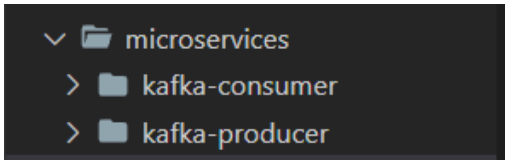


Figura 3.12 Estructura proyecto *microservices* Kafka

La estructura del consumer es bastante sencilla, teniendo únicamente dos archivos. Uno es un script Python que se suscribe al topic de recetas de Kafka y se conecta con la base de datos para almacenar la información leída [32]. El otro es un archivo *requirements* que incluye las librerías empleadas por el script. La ejecución de este fichero instala todos los paquetes contenidos en él, facilitando

la configuración del entorno y evitando tener que realizar sucesivos comandos [33].

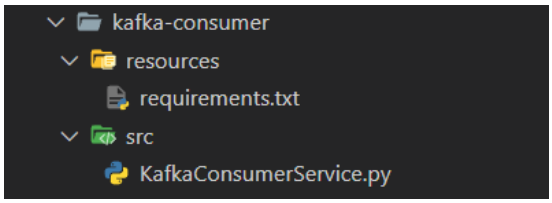


Figura 3.13 Estructura *Kafka-consumer*

El producer, por otra parte, cuenta con una estructura más compleja. Se ha implementado un proyecto SpringBoot Java, apoyado en Maven para la gestión de dependencias y del ciclo de vida de la aplicación. La clase *KafkaProducerService.java* incluye la lógica que crea el producer y realiza las peticiones de las recetas a *Tasty*. Esta clase es llamada desde el método *main()* en *KafkaProducerApplication.java*. El resto de clases son archivos de configuración de parámetros de la ejecución y de los mensajes enviados a través de Kafka.

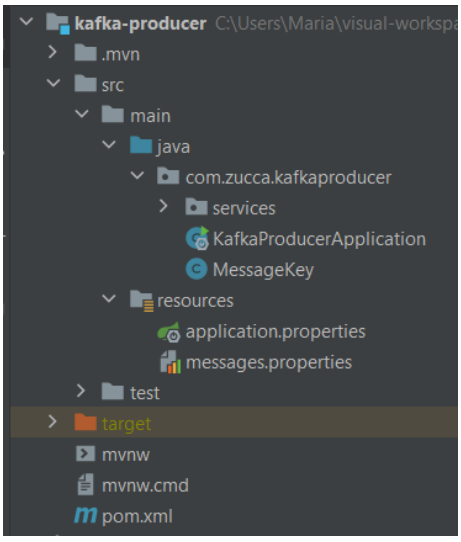


Figura 3.14 Estructura *Kafka-producer*

3.2.2.6 DISEÑO DETALLADO

La base de datos del sistema es el punto en el que se almacena toda la información recopilada por los servicios Kafka.

Al tratarse de un sistema de almacenamiento MongoDB, la estructura de la información se distribuye en colecciones y documentos BSON [34]. Cada colección es un conjunto de documentos, y cada documento se puede definir como un conjunto ordenado de claves y valores.

Para este proyecto, se han creado tres colecciones:

- **Compilaciones**: cada documento se corresponde a una categoría que engloba varias recetas relacionadas. Incluye la clave "_id" generada por MongoDB, la clave "compilation" cuyo valor es el nombre de la categoría, y la clave "recipes", la cual es un array que incluye los "id" de las recetas que forman esa compilación.

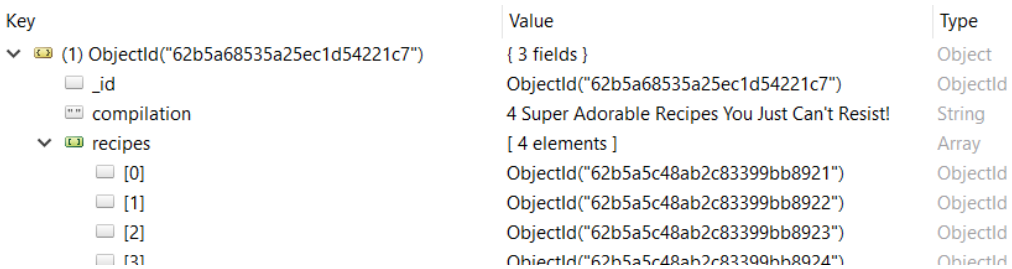


Figura 3.15 Ejemplo documento de una compilación

- **Recetas**: cada documento se corresponde a una receta. Incluye la clave "_id" generada por MongoDB, y algunos de los principales campos que se usan en este proyecto son el nombre, los ingredientes, la imagen, el tiempo de preparación… Aunque, como se observa en la imagen siguiente, incluye una gran cantidad de claves a mayores. Se ha decidido almacenarlos todos para tenerlos disponibles en futuras versiones de la aplicación, en las que serán útiles para nueva funcionalidad. Más adelante se detallará cómo se realizan las peticiones para no solicitar todos estos campos que no se van a necesitar y optimizar el proceso.

Key	Value	Type
(1) ObjectId("62b4cfbfa9ad5d5888f8dec9")	{ 51 fields }	Object
_id	ObjectId("62b4cfbfa9ad5d5888f8dec9")	ObjectId
nutrition_visibility	auto	String
country	US	String
instructions	[7 elements]	Array
keywords		String
facebook_posts	[0 elements]	Array
language	eng	String
user_ratings	{ 3 fields }	Object
id	8270	Int32
brand	null	Null
slug	omelet-with-salmon-cream-cheese	String
show_id	17	Int32
servings_noun_singular	serving	String
prep_time_minutes	null	Null
sections	[1 element]	Array
brand_id	null	Null
tags	[5 elements]	Array
nutrition	{ 0 fields }	Object
compilations	[0 elements]	Array
name	Omelet With Salmon & Cream Cheese	String
num_servings	2	Int32
buzz_id	null	Null
tips_and_ratings_enabled	true	Boolean
aspect_ratio	16:9	String
show	{ 2 fields }	Object
description		String
created_at	1654971329	Int32
draft_status	published	String
inspired_by_url	null	Null
thumbnail_url	https://img.buzzfeed.com/tasty-app-user-assets-prod-us...	String
thumbnail_alt_text		String
total_time_minutes	null	Null
video_url	null	Null
updated_at	1656012663	Int32
credits	[1 element]	Array
approved_at	1656012663	Int32
is_one_top	false	Boolean
servings_noun_plural	servings	String
renditions	[0 elements]	Array
beauty_url	null	Null
is_shoppable	true	Boolean
topics	[0 elements]	Array
total_time_tier	null	Null
video_ad_content	null	Null
seo_title		String
original_video_url	null	Null
yields	Servings: 2	String
canonical_id	recipe:8270	String
cook_time_minutes	null	Null
video_id	null	Null
promotion	full	String

Figura 3.16 Ejemplo documento de una receta

- **Usuarios:** cada documento se corresponde con un usuario del sistema. Incluye la clave "_id" generada por MongoDB, la clave "username" con su nombre de usuario, su "email", su "password", su "profilePicture", y su

array de “favorites” con los ids de las recetas que ha almacenado como favoritas. La contraseña se guarda encriptada por seguridad el usuario. Dispone de otros campos generados por MongoDB como la fecha de creación y de actualización del documento.

Key	Value	Type
(1) ObjectId("62acd3beddcda5107c373513")	{ 9 fields }	Object
_id	ObjectId("62acd3beddcda5107c373513")	ObjectId
username	maria	String
email	maria@maria.com	String
password	$2b$10$A4c4kdwQyy8jxaSfK2cV/.FNozRSm0Wh7pciVOdFqJEHibhUvj8Yy	String
profilePicture	https://images.pexels.com/photos/704569/pexels-photo-704569.jpeg?...	String
favorites	[10 elements]	Array
[0]	ObjectId("62b0932af8fa1ff8601c3325")	ObjectId
[1]	ObjectId("62b0932af8fa1ff8601c3315")	ObjectId
[2]	ObjectId("62b0932af8fa1ff8601c3316")	ObjectId
[3]	ObjectId("62b0932af8fa1ff8601c331a")	ObjectId
[4]	ObjectId("62b0932af8fa1ff8601c3313")	ObjectId
[5]	ObjectId("62b0932af8fa1ff8601c3321")	ObjectId
[6]	ObjectId("62b0932af8fa1ff8601c3312")	ObjectId
[7]	ObjectId("62b4cfbfa9ad5d5888f8dec9")	ObjectId
[8]	ObjectId("62b4d152b2d9fb8a6795e329")	ObjectId
[9]	ObjectId("62b4cfbfa9ad5d5888f8deca")	ObjectId
createdAt	2022-06-17 19:19:26.281Z	Date
updatedAt	2022-06-28 13:52:34.150Z	Date

Figura 3.17. Ejemplo documento de un usuario

Para cada documento de cada colección, su clave primaria es el “_id” generado por MongoDB, ya que así se asegura una identificación inequívoca del documento.

Para relacionar objetos de la base de datos, como es el caso en las categorías y los favoritos con sus objetos recetas correspondientes, se usa el “_id” mencionado anteriormente. De esta forma, no es necesario almacenar varias veces los mismos objetos y se evita información redundante. Basta con almacenar el “_id” del objeto que se desea referenciar, y realizar una petición que encuentre el elemento con ese valor.

3.2.3 IMPLEMENTACIÓN

Una vez analizado el diseño de las diferentes partes del proyecto, es necesario tratar los aspectos importantes de su implementación. Para ello, se explicará cómo se ha llegado a los objetivos y con qué herramientas y peculiaridades de la implementación.

3.2.3.1 IMPLEMENTACIÓN DEL FRONT-END

Lo primero es crear el proyecto. Para ello, se ha utilizado el entorno de desarrollo Visual Studio Code, y se ha creado un proyecto de tipo React.js. En el fichero *package.*json se encuentran todas las dependencias del proyecto . La versión de React empleada es ^18.1.0, donde "^" implica que se actualice a cualquier nuevo lanzamiento incluido en la versión 18.X.X. Para el estilo también se ha utilizado MaterialUI con versión ^5.6.4.

El código sigue una estructura basada en componentes, que son piezas de código en las que se emplea HTML, CSS y Javascript. De esta forma, contienen tanto la lógica como la presentación de ese componente, facilitando el mantenimiento, el entendimiento y la corrección de errores.

Peticiones al back-end

Las peticiones al back-end están incluidas en funciones de React, y se realizan haciendo uso de *axios* [35]. Esta librería de Javascript permite realizar peticiones HTTP contra el servidor Node.js, y tratar la respuesta y los errores de manera sencilla. Todas las peticiones están definidas como asíncronas y con el término *await*, lo que permite que la ejecución se detenga hasta que la petición haya finalizado. Los datos devueltos son de tipo JSON.

Las funciones en las que se ejecutan las peticiones se pueden categorizar en dos tipos. El ejemplo de la figura 3.18 hace uso del *Hook* de efecto de React [36]. "*React.useEffect()"* provoca que el código contenido en su interior se ejecute la primera vez que se carga ese componente, y, a mayores, todas las veces que el parámetro introducido en su array de dependencias varíe (última línea del código de la figura 3.18). En este caso concreto, cada vez que se carga la página principal o que se produzca un cambio de usuario, se ejecuta la petición de los favoritos del usuario logueado, si es que hay uno, para mostrarle marcadas las recetas que procedan. Dentro del *".then*" se incluye lo que se desea hacer con los datos devueltos por el back-end, y en el "*.catch*" el tratamiento de errores en caso de que haya alguno.

```
React.useEffect(() => {
    if (user) {
        const fetchFavorites = async () => {
            await axios.get('/api/v1/users/get/favorites_ids/', {
                params: {
                    user_id: user._id
                }
            }).then((res) => {
                setFavorites(res.data);
            }).catch((err) => {
                setFavorites("Error fetching the user's favorites");
            })
        }
        fetchFavorites();
    }
}, [user]);
```

Figura 3.18. [frontend] Ejemplo petición de favoritos, con useEffect()

Para mostrar otro ejemplo, se ha seleccionado una petición de tipo *DELETE*. Como se observa en la figura 3.19, no hay *useEffect(),* por lo que la función solo se ejecuta cuando se la llama en el momento en el que un usuario elimina una receta de sus favoritos.

```
const removingFavorite = async (recipe) => {
    await axios.delete('/api/v1/users/delete/favorite/', {
        params: {
            user_id: user._id,
            recipe_id: recipe._id
        },
    }).then((res) => {
        setDeleted(true);
        toast("Favorite deleted!", {
            position: toast.POSITION.TOP_RIGHT,
            autoClose: 2000
        })
    }).catch((err) => {
        setFavorites("Error removing an user's favorite: " + err);
    })
}
```

Figura 3.19. [frontend] Ejemplo petición borrar un favorito, sin useEffect()

Gestión del usuario

Para la gestión del usuario que ha iniciado sesión, React ofrece la posibilidad de compartir valores entre componentes sin necesidad de enviarlos manualmente en cada ocasión. Haciendo uso de *Context,* se cuenta con el usuario logueado como una variable global, disponible en todos los componentes donde se importe este contexto [37].

Para entender el funcionamiento de cómo se asigna y almacena el usuario, se sigue el flujo del proceso en las figuras siguientes.

```
// Si la peticion se ejecuta correctamente
const res = await axios.post("/api/v1/auth/login", {
    username: userRef.current.value,
    password: passwordRef.current.value,
})
dispatch({ type: "LOGIN_SUCCESS", payload: res.data });
```

Figura 3.20. [frontend] Petición de login

La petición de tipo POST para iniciar sesión es realizada en la figura 3.20, a la que se le pasa en el body el nombre y contraseña introducidos por el usuario. Si la autenticación se realiza de manera correcta, se lanza a capas superiores el tipo de acción (“LOGIN_SUCCESS” para indicar éxito) y los datos devueltos por la petición (todos los del usuario menos la contraseña).

```
const Reducer = (state, action) => {
    switch (action.type) {
      case "LOGIN_SUCCESS":
        return {
          user: action.payload,
          isFetching: false,
          error: false,
        };
```

Figura 3.21 [frontend] Acciones del *reducer.js* ante un login válido

En la figura 3.21 se aprecia cómo se hace un switch en función del tipo de acción ejecutada. En este caso, se asigna a “*user”* los datos traídos del back-end y

enviados en el campo *payload*. También, se indica que el proceso ha terminado (“*isFetching”*) y que no hay ningún error (“*error*”).

Hooks de estado

En cada componente, se puede dar el caso de que sean necesarios estados para guardar ciertos cambios efectuados por el usuario. Para ello, React utiliza *Hooks de estados* [38]. Un ejemplo del *Hook* que controla cuándo se abre el modal con la información de la receta es la figura 3.22.

```
const [open, setOpen] = React.useState(false);
const handleOpen = () => {
    setOpen(true);
};
const handleClose = () => {
    setOpen(false);
};
```

Figura 3.22 [frontend] Fragmento de código de un *Hook* de estado

Se define una constante con la variable y el método encargado de asignarle valor. En este caso, la variable se inicializa a “*false”,* y, cuando se haga click en el componente de la receta, se ejecutará la función “*handleOpen()”*, dando valor “*true”* a la variable de estado. De la misma forma, al cerrar el modal, se ejecutará “*handleClose()”* para devolverle el estado “*false”*.

Así, el componente Modal define su estado como se ve en la figura 3.23.

```
<Modal
    style={{ display: 'flex', alignItems: 'center', justifyContent: 'center', outline: 'none' }}
    open={open}
    onClose={handleClose} className="modalRecipe"
>
```

Figura 3.23 [frontend] Fragmento de código del modal de una receta

Enrutamiento entre páginas

Para definir las rutas de la aplicación web, se hace uso de la librería *react-router-dom*, la cual incluye componentes de navegación a los cuales solo hay que especificarle la ruta y el componente que queremos que cargue. En la imagen

siguiente se muestra el enrutamiento empleado en el fichero *App.js*, donde, por defecto cargará el componente *Home*. Si se solicita alguna de las otras rutas se comprobará antes si hay un usuario autenticado, y cargará una vista u otra en función del resultado.

```
return (
  <Router>
    <Fragment>
      <Routes>
        <Route exact path="/" element={<Home />} />
        <Route path="/register" element={user ? <Home /> : <Register />} > </Route>
        <Route path="/login" element={user ? <Home /> : <Login />} > </Route>
        <Route path="/favorites" element={user ? <Favorites /> : <Login />} > </Route>
      </Routes>
    </Fragment>
  </Router>
)
```

Figura 3.18 [frontend] Enrutamiento entre las páginas

Conexión con el back-end

Para establecer a qué dirección y puerto se deben hacer las peticiones, se especifica en el "*package.json*" la directiva "proxy". De esta forma, cada petición en la que no se indique ninguna dirección, será dirigida a *localhost:5000.*

```
"proxy": "http://localhost:5000"
```

Figura 3.19 [frontend] Directica proxy

3.2.3.2 IMPLEMENTACIÓN DE LOS SERVICIOS DOCKER

El microservicio de Docker se ha desarrollado también en Visual Studio Code.

Como se especificó en el apartado de diseño, en este microservicio tenemos un archivo *docker-compose.yml,* encargado del despliegue de un contenedor para cada servicio. Uno con la imagen de MongoDB, otro de Zookeeper y otro de Kafka. Así mismo, se especifican los puertos de cada uno, y, para Kafka, el topic

de recetas a través del cual se enviarán para almacenarlas. Su contenido es el siguiente:

```yaml
version: "3.8"

services:
  mongo:
    image: mongo:5.0.3
    container_name: mongo
    hostname: mongo
    restart: unless-stopped
    ports:
      - "27017:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: malvac10
      MONGO_INITDB_ROOT_PASSWORD: malvaczucca
    volumes:
      - ./persistence/mongo/data:/data/db
      - ./persistence/mongo/log:/var/log/mongodb/

  zookeeper:
    image: wurstmeister/zookeeper
    container_name: zookeeper
    restart: unless-stopped
    ports:
      - "2181:2181"

  kafka:
    image: wurstmeister/kafka
    container_name: kafka
    restart: unless-stopped
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: 127.0.0.1
      KAFKA_CREATE_TOPICS: "recipes_topic:1:1"
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
```

Figura 3.20 [docker] Fichero docker-compose.yml

La carpeta de persistencia se genera en la ruta especificada en *volumes,* del servicio “mongo” del archivo anterior, e incluye los datos de nuestro MongoDB. Permite que, al apagar y arrancar nuestro contenedor, los datos sigan existiendo. Se ha guardado una carpeta *data* con el contenido de la base de datos y otra *log* con el registro de los logs por si fuese necesario consultarlos ante algún fallo.

3.2.3.3 IMPLEMENTACIÓN DEL BACK-END

El back-end ha sido implementado en Visual Studio Code, con Node.js y Express.js. Puesto que es un proyecto Node, contiene un *package.json* donde están las dependencias.

El primer fichero que mencionar es *config.js*. Este archivo exporta variables para poder utilizarlas de manera global en el resto del proyecto. Sus valores son el puerto de escucha, la URI para la conexión con la base de datos, y la versión de la API definida por el desarrollador [39].

```
module.exports = {
    // Puerto del servidor
    PORT: 5000,
    // La cadena de conexión con la base de datos
    DATABASE_URI: 'mongodb://malvac10:malvaczucca@localhost:27017/zucca?authSource=admin',
    // Versión actual de la API
    API_VERSION: 'v1',
};
```

Figura 3.21 [backend] Variables globales config.js

En el archivo *index.js* se establece, a través de Mongoose, una conexión con la base de datos y se indica en qué puerto debe escuchar la API las peticiones. También, se establece a dónde se va a redirigir cada petición en función de la ruta. La conexión, las rutas de esta API y la escucha en el puerto se muestran en el fragmento de código de la figura 3.24.

```
//Conexion con la base de datos
mongoose.connect(config.DATABASE_URI, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
}).then(console.log("Connected to MongoDB - database: Zucca"))
    .catch(err => console.log(err));

// Rutas
app.use("/api/v1/recipes", recipesRoute);
app.use("/api/v1/auth", authRoute);
app.use("/api/v1/users", userRoute);

// Desplegar en el puerto especificado
app.listen(PORT, () => {
    console.log('Server on port', PORT);
});
```

Figura 3.22 [backend] Fragmento del fichero index.js

Para explicar de manera visual un ejemplo de una petición, se ha escogido un fragmento de código que representa la acción de crear un nuevo usuario. Esta es ejecutada cuando un usuario, desde el navegador, se registra en la página web. El servidor recibe los datos del usuario a través del body, ya que se trata de una petición POST [40]. Este, cifra la contraseña y pasa a crear un objeto usuario con la información recibida para posteriormente almacenarlo en la base datos y devolverlo.

```
// Registrarse
router.post("/register", async (req, res) => {
    try {
        const salt = await bcrypt.genSalt(10);
        const hashedPassword = await bcrypt.hash(req.body.password, salt);

        const newUser = new User({
            username: req.body.username,
            email: req.body.email,
            password: hashedPassword,
        })
        const user = await newUser.save();
        res.status(200).json(user)
    } catch (err) {
        res.status(500).json(err)
    }
});
```

Figura 3.23 [backend] Ejemplo de petición Registrarse

Lo que se muestra a continuación es el modelo de un usuario, incluido en la carpeta "*models".* En él se hallan sus campos y si son obligatorios u opcionales. Cuando se crea un usuario se devuelve el objeto con todos sus campos menos la contraseña debido a razones de seguridad.

```
import { Schema, model } from 'mongoose';

const UserSchema = new Schema({
    username: {
        type: String,
        required: true,
        unique: true,
    },
    email: {
        type: String,
        required: true,
        unique: true,
    },
    password: {
        type: String,
        required: true,
    },
    profilePicture: {
        type: String,
        default: "",
    },
    favorites: [{
        type: Schema.Types.ObjectId,
    }],
},
    { timestamps: true }
);

export default model('User', { password, ...others } = UserSchema);
```

Figura 3.24 [backend] Modelo de un User

También, es importante destacar que a la hora de recibir peticiones de recetas no se devuelven todos los campos almacenados en el sistema de almacenamiento. Como se trató en apartados anteriores, solo se devuelven los campos necesarios. Esto se consigue haciendo uso de las proyecciones de MongoDB, las cuales te permiten obtener los campos especificados y ahorrar recursos de traer información que no se va a utilizar.

En la figura 3.25 se define la variable con los campos de la proyección que se desean obtener, y en la 3.26 se hace la petición pasándole esta variable a la función "project()".

```
const projection = {
    instructions: 1, name: 1, description: 1,
    sections: 1, thumbnail_url: 1, country: 1,
    total_time_minutes: 1, total_time_tier: 1
};
```

Figura 3.25 [backend] Variable usada para una proyección

```
collection.find({ name: { $regex: regex } }).project(projection)
    .skip(skip_page).limit(PAGE_SIZE).toArray((err, data) => {
        res.send(JSON.stringify(data)).status(200);
    });
```

Figura 3.26 [backend] Petición con uso de proyecciones

3.2.3.4 IMPLEMENTACIÓN DE LOS SERVICIOS KAFKA

Los servicios Kafka incluyen el Kafka Producer y el Kafka Consumer. Para que el Consumer sea capaz de leer los mensajes publicados por el Producer, tiene que estar suscrito al *topic* correspondiente. Al desplegar por primera vez el Docker que contiene a Kafka, se creó un *topic* llamado “recipesZucca”, y se ha especificado que el Producer publique y que el Consumer se suscriba a este flujo de mensajes.

El **Kafka Producer** ha sido desarrollado en IntelliJ IDEA 2021.3.2. Puesto que se trata de un proyecto SpringBoot de Java con Maven, este entorno de desarrollo ofrece facilidades para proyectos programados en este lenguaje.

En la figura 3.27 se muestra el método Java que se encarga de realizar las peticiones a la API *tasty* para obtener y publicar las recetas en el topic de recetas de Kafka.

```java
@Scheduled(fixedDelayString = "${fixedDelay.in.milliseconds}")
public void sendMessageToKafka() {
    String from = "0", size = "40";
    OkHttpClient client = new OkHttpClient();

    for (int i = 0; i < 150; i++) {
        Request request = new Request.Builder()
                .url("https://tasty.p.rapidapi.com/recipes/list?from=" + from + "&size=" + size)
                .get()
                .addHeader( name: "X-RapidAPI-Host",  value: "tasty.p.rapidapi.com")
                .addHeader( name: "X-RapidAPI-Key",  value: "efe6594a81msh8d7ff805d6c89f8p112ae0jsn8dd67145ab06")
                .build();
        try {
            Response response = client.newCall(request).execute();
            if (response.body() != null) {
                String message = response.body().string();
                JSONObject jsonObject = new JSONObject(message);
                for (int j = 0; j < jsonObject.getJSONArray( key: "results").length(); j++) {
                    JSONObject recipe = jsonObject.getJSONArray( key: "results").getJSONObject(j);
                    kafkaTemplate.send(topic, recipe.toString());
                    LOGGER.info(messageSource.getMessage( code: "message.sent",
                            new String[]{KafkaProducerService.class.getSimpleName(), recipe.getString( key: "name")},
                            LocaleContextHolder.getLocale()));
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        from = String.valueOf(Integer.parseInt(from) + 40);
    }
}
```

Figura 3.27 [kafka] Fragmento que pide y publica recetas en el topic

Con la librería *OkHttpClient* se crea un cliente para realizar la petición. Todo el proceso de las peticiones y publicaciones se realiza dentro de un bucle *for* que se ejecuta 150 veces. Cada una de ellas, emplea las variables *from* y *size* para ir pidiendo recetas de 40 en 40, ya que *tasty* consta de paginación que no permite obtener más de 40 recetas por petición. De esta forma, cada respuesta obtenida se convierte a un objeto JSON que se recorrerá como array, para transmitir uno a uno los resultados del conjunto de recetas en el topic al que está suscrito el Kafka Consumer. Este producer genera un total de 6000 documentos.

Los parámetros de configuración del *topic*, dirección del servidor y puerto, entre otros, se especifican en el archivo de configuración “application.properties”.

```
#Config Kafka
kafka.topic = recipesZucca
spring.kafka.producer.bootstrap-servers = 127.0.0.1:9092
```

Figura 3.28 [kafka] Archivo "application.properties" del Kafka Producer

El **Kafka Consumer** se ha implementado en PyCharm 2021.1.3, ya que el lenguaje en el que está programado es Python. Como librerías se utiliza k*afka-python*, *python-snappy* y *pymongo* [41]–[43]. Estas están incluidas en el fichero *requirements* para su instalación con solo un comando, como se especificó en el apartado de diseño.

- k*afka-python* permite el uso de interfaces para poder crear un consumer suscrito a un *topic*, leer los mensajes, y cualquier tarea relacionada con este sistema de procesamiento de flujo de datos.
- *python-snappy* es una librería de compresión y descompresión. Su instalación es necesaria para leer correctamente los datos de Kafka, que están comprimidos en formato *snappy*.
- *pymongo*, por otro lado, contiene las herramientas para trabajar desde Python con MongoDB.

Lo principal que tiene este servicio es el método "data_parser()", el cual genera un Consumer al que se le especifica el *topic* al que se debe suscribir y la dirección del servidor donde debe leer los mensajes. En este caso, "*recipesZucca*" y "localhost:9092", respectivamente.

Este consumer automáticamente procesa cada mensaje transmitido en ese *topic*, y lo recoge en un bucle donde cargará la información en JSON y empezará a evaluar si se trata de una receta o una categoría de recetas. Para ello, se comprueba el texto de uno de los campos de estos objetos, donde si incluye la palabra "recipe" es una receta, y si incluye "compilation" es una categoría. En ambos casos, se comprueba antes que no exista ya esa receta en la base de datos para no insertar duplicados.

```python
def data_parser():
    consumer = KafkaConsumer('recipesZucca', bootstrap_servers=['localhost:9092'])

    for msg in consumer:
        recipe = json.loads(msg.value)
        canonical_id = str(recipe['canonical_id'])
        try:
            if "recipe" in canonical_id:
                same_recipe = db.recipes.find_one({'canonical_id': recipe['canonical_id']})
                # Si no existe la receta:
                if same_recipe == None:
                    db.recipes.insert_one(recipe)
            elif "compilation" in canonical_id:
                # Se crea un documento para esa categoría
                document = {"compilation": recipe['name'], "recipes": []}
                # Se recorre cada receta de la  categoría
                for element in recipe['recipes']:
                    same_recipe = db.recipes.find_one({'canonical_id': element['canonical_id']})
                    # Si no existe la receta:
                    if same_recipe == None:
                        # Si ya existe el campo _id tiene que ser eliminado
                        if '_id' in element:
                            del element['_id']
                        data_inserted = db.recipes.insert_one(element)
                        document['recipes'].append(data_inserted.inserted_id)
                    else:
                        document['recipes'].append(same_recipe['_id'])
                db.compilations.insert_one(document)
```

Figura 3.29 [kafka] Consumer de Kafka

Como se puede ver en la figura 3.29:

- Si se trata de una receta se inserta como un documento nuevo.
- Si se trata de una categoría, se recorre cada receta de esa categoría y se inserta como un documento nuevo. En algunos casos, *tasty* devuelve estas recetas con el *"_id"* ya existente, lo cual provoca errores inesperados ya que este campo siempre debería ser autogenerado por MongoDB en el momento de la inserción. Para evitarlos, se elimina del objeto. También, se inserta la categoría con su nombre y los "*_id"* de cada receta.

3.2.4 PRUEBAS

En todo proyecto software es necesario realizar una serie de pruebas que demuestren que el sistema desarrollado cumple con lo especificado. Se han desarrollado pruebas de sistema y pruebas de validación *alfa* sobre la página web.

La tabla 3.40 muestra cada prueba de sistema realizada sobre la arquitectura de datos, con un identificador, una descripción explicativa y el resultado obtenido.

Tabla 3.40 Pruebas de sistema en el sistema de datos

PRUEBAS DE SISTEMA – SISTEMA DE DATOS		
Id	Descripción	Salida
PS01	Se ejecuta el comando “docker-compose up docker-compose.yml” en el directorio “docker-services”	Contenedores levantados
PS02	Se levanta el Docker Compose cuando alguno de los puertos está ocupado	Mensaje de error, colisión de puertos
PS03	Se ejecuta la clase Java que tiene el método “main()” en el Kafka Producer con la API *tasty* y los contenedores en funcionamiento	Se despliega el Producer y se ejecuta correctamente
PS04	Se ejecuta la clase Java que tiene el método “main()” en el Kafka Producer con la API *tasty* inoperativa	Se despliega el Producer y se muestran mensajes de error informando de peticiones fallidas
PS05	Se ejecuta la clase Java que tiene el método “main()” en el Kafka Producer con el/los contenedor/es de Kafka y/o ZooKeeper sin levantar	Mensaje de error alertando de que no puede conectarse con Kafka
PS06	Se ejecuta el script Python del Kafka Consumer con los contenedores levantados	Se conecta a Kafka y a la base de datos
PS07	Se ejecuta el script Python del Kafka Consumer con algún contenedor sin levantar	Mensaje de error de conexión con Kafka,

		ZooKeeper y/o MongoDB
PS08	Se ejecuta el script Python del Kafka Consumer mientras el Producer está publicando mensajes	El Producer lee correctamente los mensajes
PS09	Se ejecuta el script Python del Kafka Consumer cuando el Producer ya ha publicado los mensajes	El Producer lee correctamente los mensajes
PS10	Se ejecuta el back-end sin levantar la base de datos	Mensaje de error de conexión con la base de datos
PS11	Se ejecuta el back-end con la base de datos levantada	Conexión establecida y servidor activo en el puerto 5000
PS12	Se realizan peticiones al back-end cuando no está levantado	Mensajes de error avisando de que el servidor no está disponible
PS13	Se realizan peticiones al back-end sin pasarle los parámetros necesarios (en función de cada petición)	Mensajes de error avisando de que no se han recibido los parámetros esperados
PS14	Se realizan peticiones al back-end pasándole los parámetros necesarios (en función de cada petición)	La petición se ejecuta correctamente y devuelve los datos establecidos
PS15	Se realizan peticiones al back-end pasándole parámetros inválidos	Mensaje de error avisando de que no se ha podido ejecutar la petición

PS16	Se realizan peticiones al back-end a una ruta que no está definida	Mensaje de error de que no se ha podido ejecutar la petición

La tabla 3.41 muestra cada prueba de validación realizada sobre la aplicación web, con un identificador, una descripción explicativa y el resultado obtenido.

Tabla 3.41 Pruebas de validación en la página web

PRUEBAS DE VALIDACIÓN – PÁGINA WEB		
Id	Descripción	Salida
PV01	Se accede a la dirección que aloja la web sin haber iniciado sesión	Página principal de la aplicación web sin usuario
PV02	Se accede a la dirección que aloja la web habiendo iniciado sesión	Página principal de la aplicación web con usuario
PV03	Se introduce la URL de la página de favoritos sin haber iniciado sesión	Redirección a la página de login
PV04	Se intenta añadir un favorito sin iniciar sesión	Mensaje de aviso para que inicie sesión
PV05	Click en una receta	Se despliega la ventana emergente con su información
PV06	Click en una categoría que no está seleccionada	Se muestran únicamente recetas de esa categoría
PV07	Click en una categoría ya seleccionada	Se vuelven a mostrar todas las recetas de la aplicación

PV08	Click en otra categoría cuando una está seleccionada	Se cambia la categoría seleccionada
PV09	Click en el logo de la aplicación desde cualquier vista	Redirección a la página principal
PV10	Se introduce texto en la barra de búsqueda y se hace click en el icono de la lupa o se pulsa la tecla “ENTER” en el teclado	Se muestran únicamente recetas relacionadas con esa búsqueda
PV11	Se vacía el texto de la barra de búsqueda y se hace click en el icono de la lupa o se pulsa la tecla “ENTER” en el teclado	Se vuelven a mostrar todas las recetas de la aplicación
PV12	Se hace click en el enlace de “LOGIN“ cuando el usuario no está logueado	Redirección a la página de login
PV13	Se hace click en el enlace de “REGISTER” cuando el usuario no está logueado	Redirección a la página de registro
PV14	Se hace click en el botón de “REGISTER” desde la página de login	Redirección a la página de registro
PV15	Se hace click en el botón de “LOGIN” desde la página de registro	Redirección a la página de login
PV16	Se introducen un nombre de usuario y una contraseña válidos en la página de login y se hace click en el botón de “LOGIN”	Redirección a la página principal con sesión iniciada
PV17	Se introducen un nombre de usuario y una contraseña incorrectas en la página de login y se hace click en el botón de “LOGIN”	Mensaje de alerta de credenciales incorrectas
PV18	Se deja vacío alguno de los campos del formulario de login y se hace click en el botón de “LOGIN”	Mensaje de alerta avisando de que debe rellenar los campos

PV19	Se introducen un nombre de usuario que no exista, email y contraseña en la página de registro y se hace click en el botón de “REGISTER”	Redirección a la página de login para que el usuario inicie sesión
PV20	Se introduce un nombre de usuario existente, un email y una contraseña en la página de registro y se hace click en el botón de “REGISTER”	Mensaje de alerta avisando de que ya existe un usuario con ese nombre
PV21	Se deja vacío alguno de los campos del formulario de registro y se hace click en el botón de “REGISTER”	Mensaje de alerta avisando de que debe rellenar los campos
PV22	Estando logueado se hace click en la imagen del usuario y se pincha en la primera opción “Favorites”	Redirección a la página de favoritos del usuario
PV23	Estando logueado se hace click en la imagen del usuario y se pincha en la segunda opción “Logout”	Redirección a la página de login y cierre de la sesión del usuario
PV24	Click en otro número de página en el componente de paginado que se encuentra al final de la página web	Carga de otras 40 recetas en función del número de página
PV25	Click en la flecha “>” o “<” en el componente de paginado que se encuentra al final de la página web	Carga de otras 40 recetas en función de la página que toque
PV26	Se intenta añadir un favorito estando logueado	Se añade correctamente
PV27	Se hace click en el icono de favorito de una receta que ya está añadida como favorita desde la página principal	Se elimina de los favoritos de ese usuario

PV28	Se hace click en el icono de favorito de una receta que ya está añadida como favorita desde la página de favoritos	Se elimina de los favoritos de ese usuario
PV29	Se pulsa la tecla “ESC” cuando el modal de una receta se encuentra abierto	Se cierra el modal
PV30	Se pulsa fuera del modal cuando el modal de una receta se encuentra abierto	Se cierra el modal

3.3 EL PRODUCTO DEL DESARROLLO

A continuación, se muestra el producto final, explicando con la ayuda de imágenes el funcionamiento del mismo.

3.3.1 SISTEMA DE DATOS

En la arquitectura de recopilación automática, almacenamiento y gestión de la información, el usuario cuenta con los cuatro microservicios tratados a lo largo de este documento.

Lo primero que ejecuta es el servicio de Docker para crear los contenedores que incluirán las imágenes de MongoDB, Kafka y ZooKeeper.

Figura A.1 Contenedores de Docker

Una vez disponga de ellos y estén levantados, puede ejecutar el Kafka Consumer y el Kafka Producer para llenar la base de datos con las recetas.

En la figura A.2 se pueden observar los mensajes enviados por el Kafka Producer, los cuales se imprimen para comprobar que se está ejecutando correctamente.

Figura A.2 Logs de mensajes enviados por Kafka Producer

Cuando termine la ejecución de estos servicios, su base de datos dispondrá de las colecciones “compilations” y “recipes”. La colección “users” será creada en el momento en el que se inserte por primera vez un usuario.

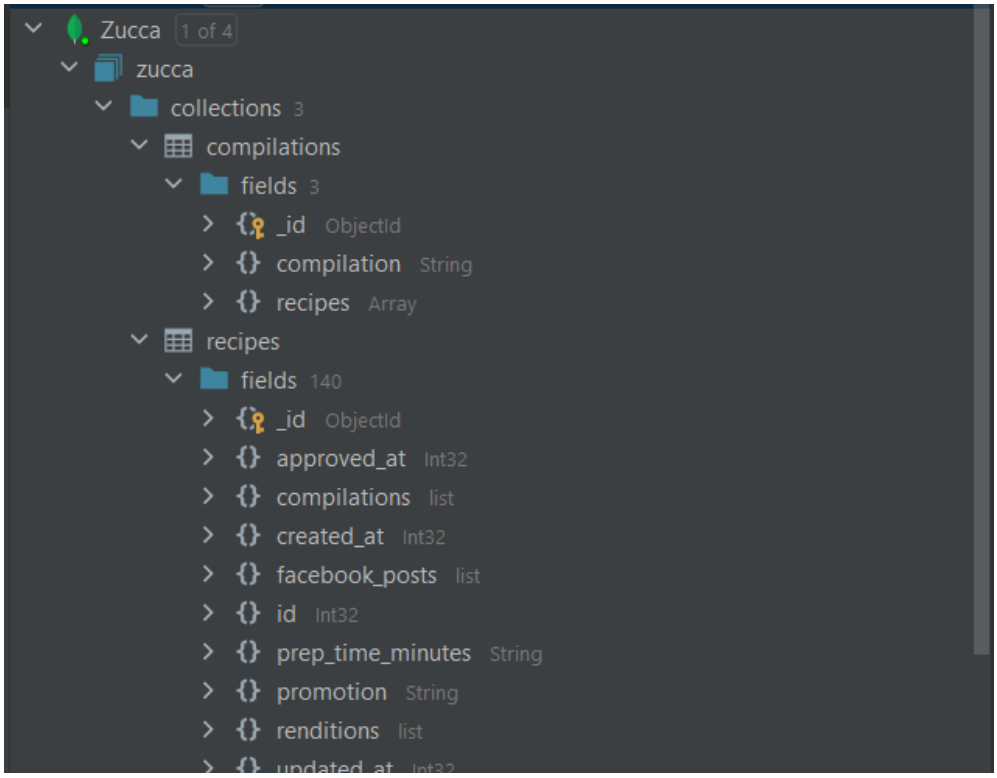


Figura A.3 Base de datos después de ejecutar los servicios Kafka

En este punto, el usuario puede o bien desarrollar su propio back-end con sus peticiones personalizadas, o emplear el ya existente.

Si escoge utilizar el que se ha desarrollado en este proyecto, dispone de las siguientes peticiones:

- Recetas
    - Obtener el número total de recetas, comprobando si se desea contar el de las correspondientes a una categoría, a una búsqueda o a todas las disponibles.
    - Obtener todas las recetas.
    - Obtener recetas coincidentes con el nombre pasado como parámetro.
    - Obtener recetas de una categoría cuyo id se pasa como parámetro.
    - Obtener receta coincidente con el id pasado como parámetro.
- Usuarios
    - Obtener usuario cuyo id es pasado como parámetro.
    - Añadir el id de una receta al array de favoritos de un usuario, cuyos id's son pasados como parámetros.

    - Eliminar una receta del array de favoritos del usuario cuyos id's son pasados como parámetros.
    - Obtener las recetas del array de favoritos del usuario cuyo id es pasado como parámetro.
    - Obtener los id's de las recetas del array de favoritos del usuario cuyo id es pasado como parámetro
- Autenticación
    - Añadir un nuevo usuario con los datos pasados como parámetros, y devolverlo.
    - Devolver el usuario que coincida con las credenciales pasadas como parámetros (se devuelven todos los datos menos su contraseña).

Ya se ha tratado en otros capítulos que los campos que se obtienen de cada receta están limitados a los que se consideran necesarios para la aplicación web Zucca. Si se desea obtener más campos de cada receta lo único que habría que hacer es consultar en la base de datos cuáles están disponibles y añadirlos en la proyección que se usa en las peticiones.

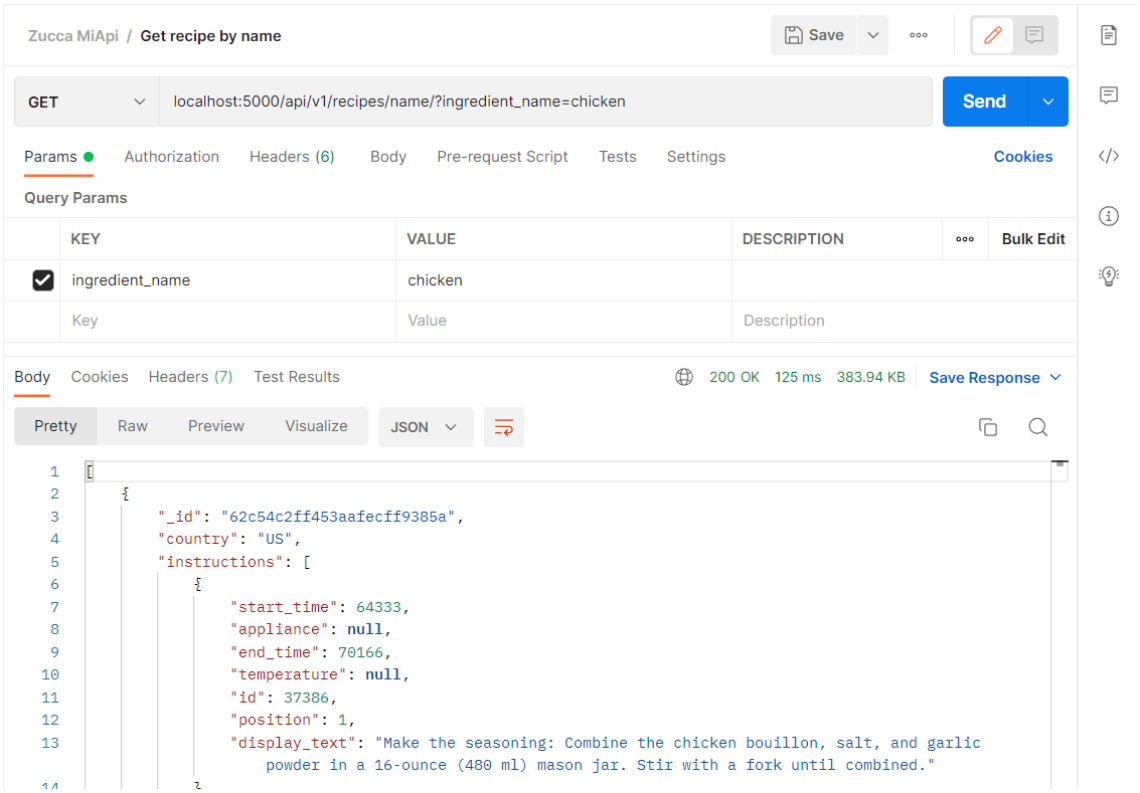


Figura A.4 Petición de recetas de "*chicken*" desde PostMan

En la figura A.4 se puede observar un ejemplo de respuesta de petición hecha desde PostMan, donde se solicitan las recetas que lleven "*chicken*" incluido en el título.

3.3.2 PÁGINA WEB

Lo primero que se encuentra un usuario al acceder a la página web es la página principal. En ella, se ve un componente superior *header* donde es posible realizar búsquedas de recetas, iniciar sesión o registrarse.

Si se observa el componente de debajo podrá ver una imagen que se corresponde con la marca, así como con el texto "Cooking recipes" y el nombre de la aplicación web.

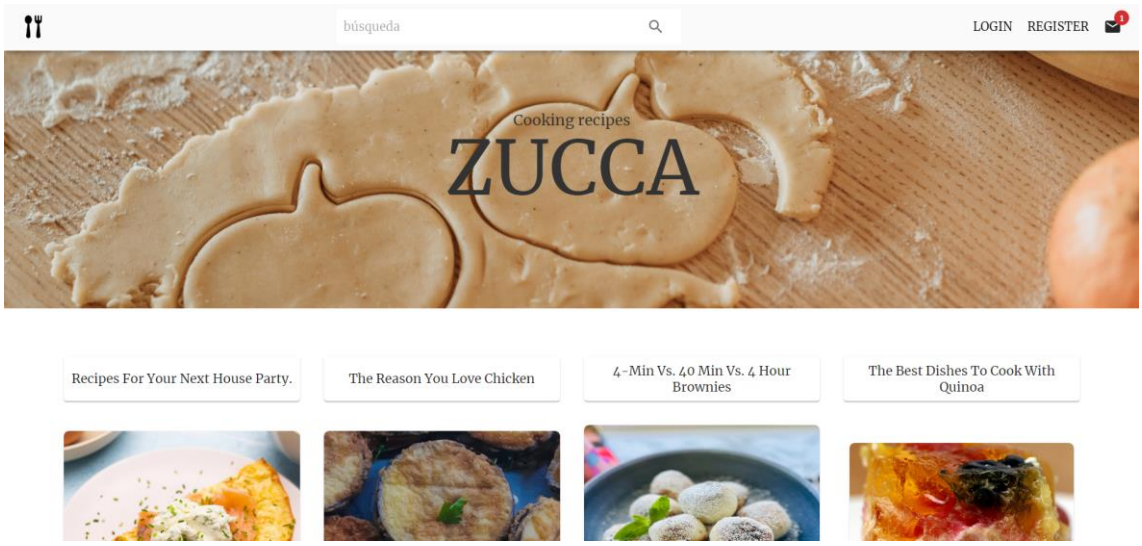


Figura A.5 Página principal sin iniciar sesión

Antes de visualizar las recetas, cuenta con cuatro menús de categorías que también se cargan aleatoriamente al entrar. Si se hace click en alguna de ellas, se le mostrarán sus recetas correspondientes.

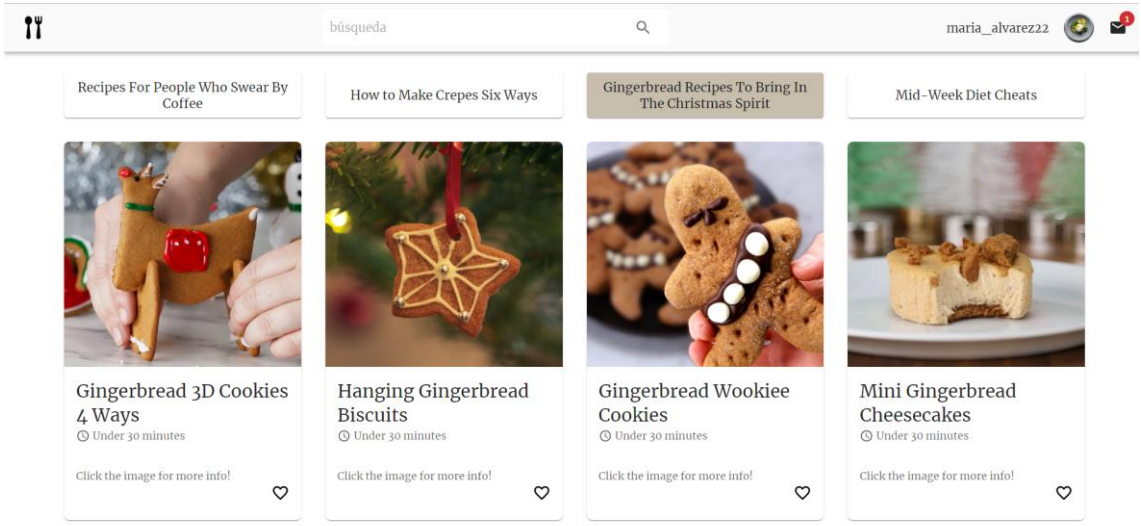


Figura A.6 Categoría seleccionada en la página principal

Desplazándose un poco hacia abajo, observará diferentes recetas que se cargan aleatoriamente al abrir la página. En ellas, se muestra una imagen de la misma, el nombre, el tiempo estimado de preparación, su descripción en caso de que

tenga y un icono de corazón para marcarlas como favoritas. Si se intenta añadir una como favorita le saldrá una alerta emergente con duración de 5 segundos avisando de que debe iniciar sesión.

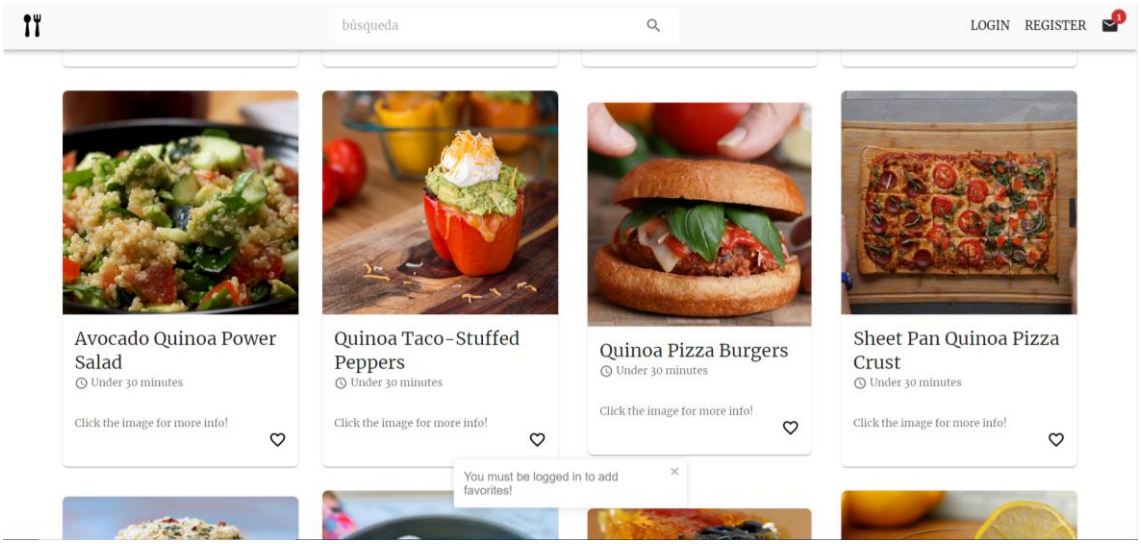


Figura A.7 Recetas en la página principal

Al final de la página, también se cuenta con un elemento para recorrer las diferentes páginas de recetas. El número de páginas varía en función del número de recetas disponibles. A su vez, en la esquina inferior derecha está disponible la dirección de correo del administrador, por si se desease contactar.

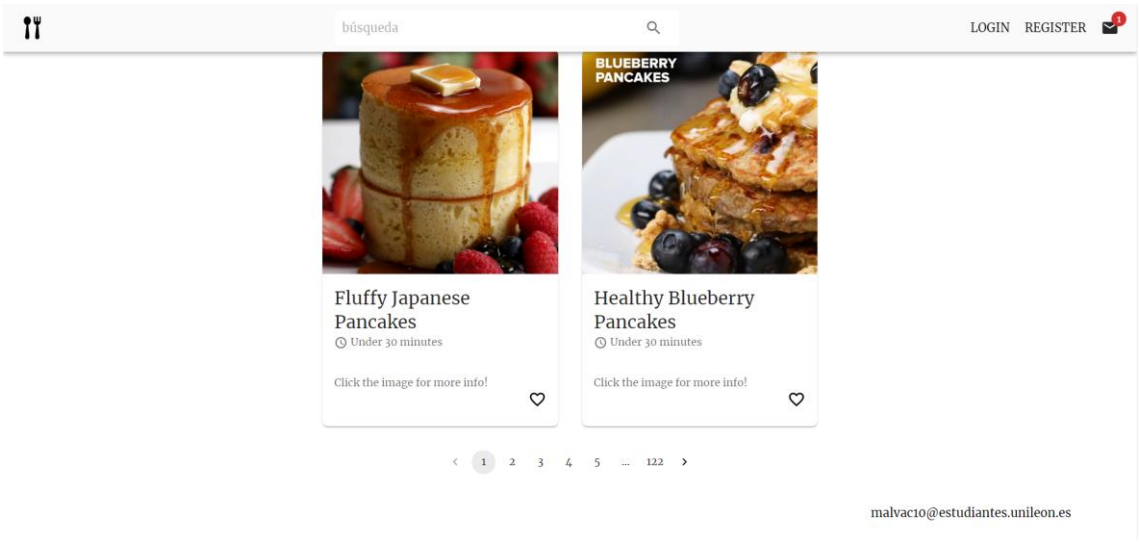


Figura A.8 Parte final de la página principal

Si hace click en una de estas recetas, se despliega una ventana emergente con información detallada sobre cómo preparar la receta (ingredientes, medidas y pasos). Para cerrarla se puede hacer click fuera de la ventana, o pulsar la tecla “Esc” del teclado.

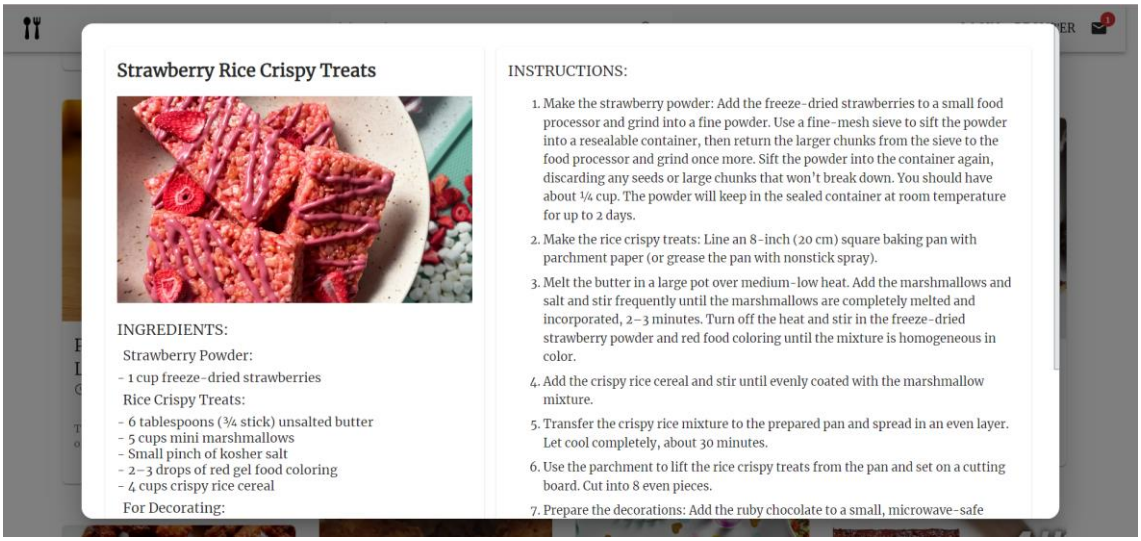


Figura A.9 Información de una receta

Si el usuario accede a la página de registrarse, se le mostrará la siguiente ventana con un formulario donde introducir el usuario, email y contraseña de la

cuenta que desea crear. Si hay algún error, como que el nombre de usuario ya está registrado en el sistema, se le mostrará la alerta correspondiente.

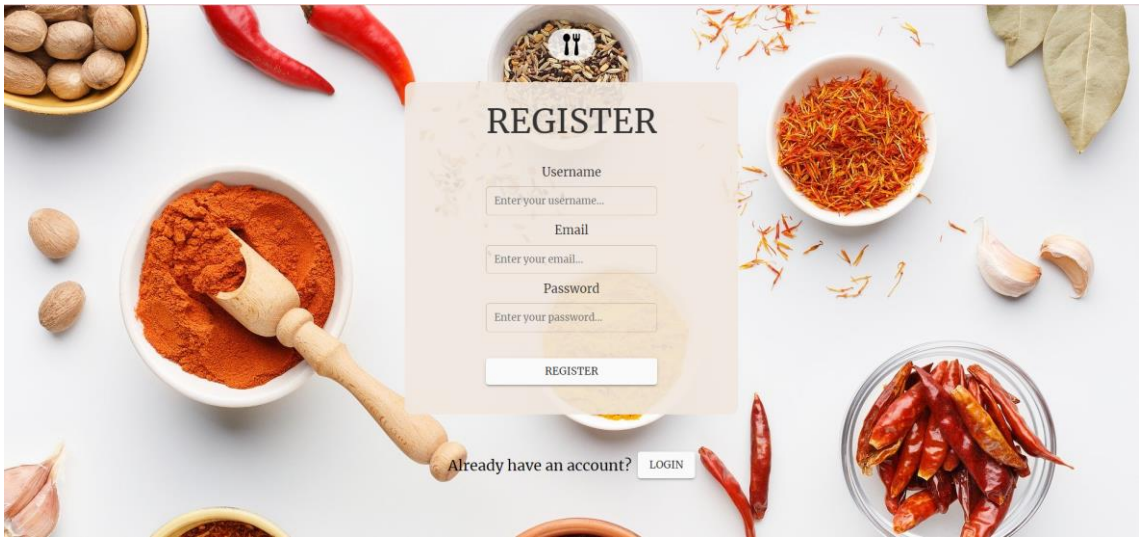


Figura A.10 Página de registrarse

Por otro lado, si ya tiene una cuenta puede acceder a la página de login. En ella, consta de otro formulario donde introducir sus datos para iniciar sesión. En caso de que el nombre de usuario o contraseña no sean correctas, se le mostrará un mensaje de error para que lo vuelva a intentar.

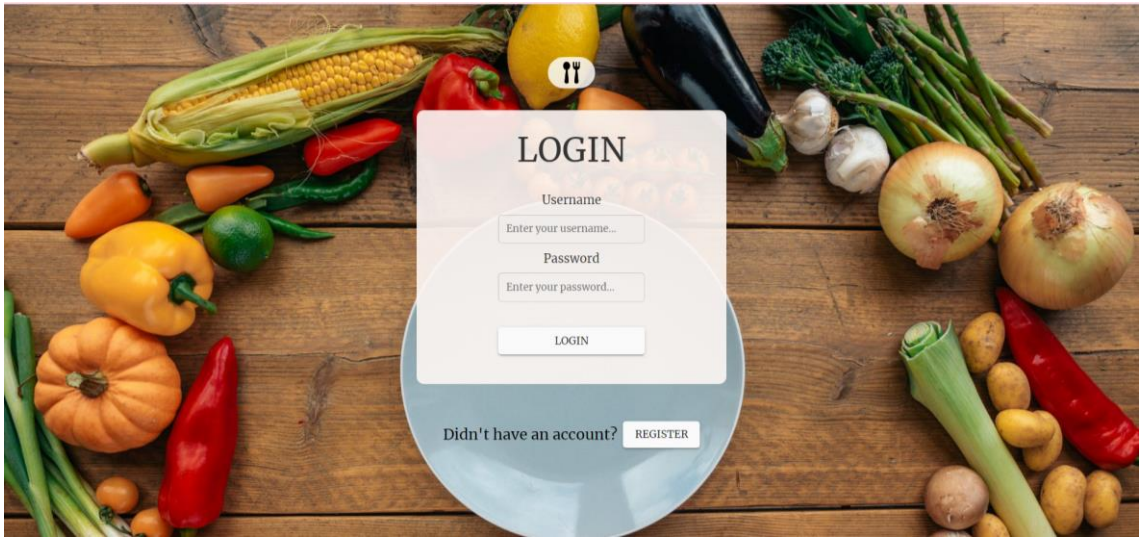


Figura A.11 Página de login

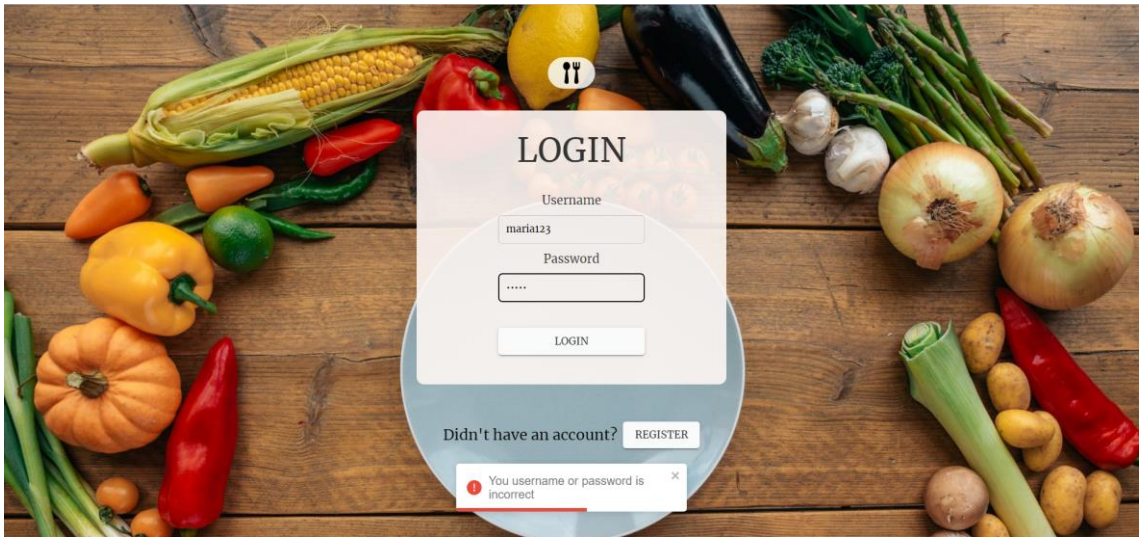


Figura A.12 Alerta de credenciales incorrectas

Cuando lleva a cabo un login exitoso es redirigido de nuevo a la página principal, pero esta vez en vez de los enlaces para acceder a la página de login o de registro se le mostrará un icono de su perfil para acceder a sus favoritos o cerrar sesión.

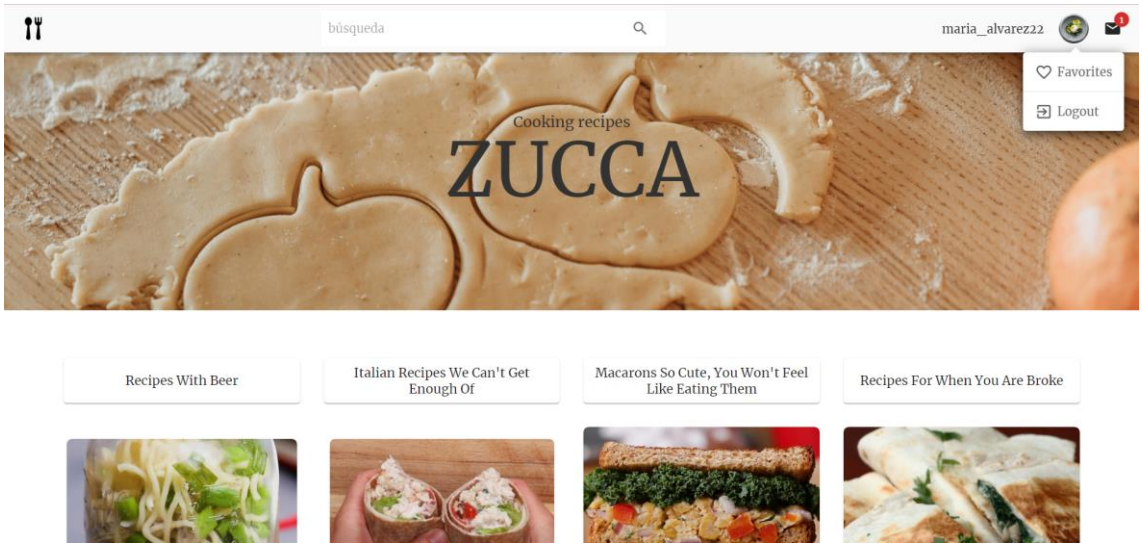


Figura 3.1 Menú del usuario desplegado

Así mismo, las recetas que se muestran en el componente central aparecerán con un icono de corazón rojo en caso de que estén marcadas como favoritas, y negro en caso de que no.

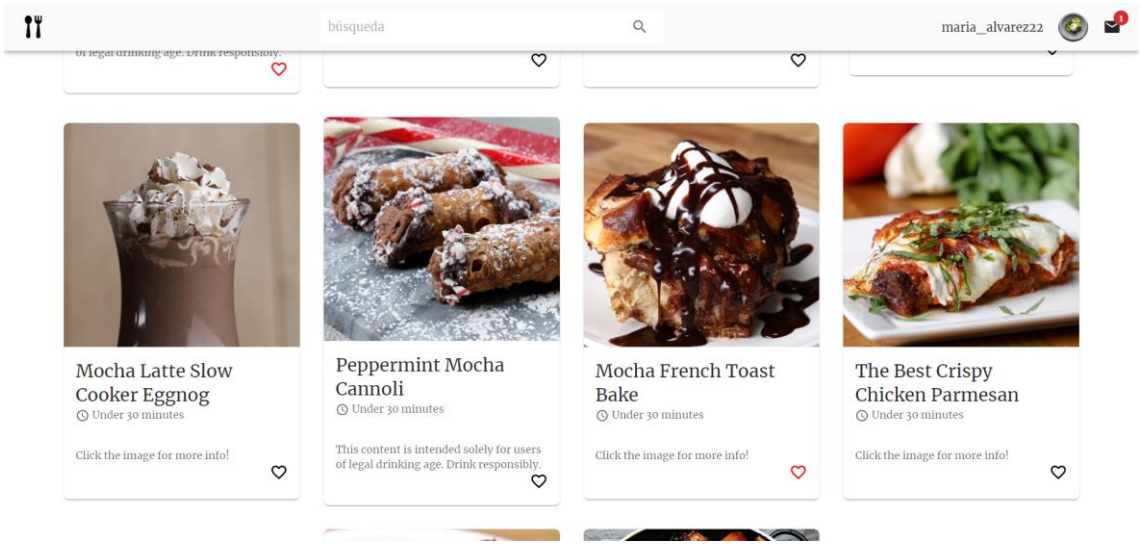


Figura A.13 Recetas cuando el usuario ha iniciado sesión

Por último, el usuario puede acceder a su página personal de favoritos, donde aparecerán en el mismo formato que en la página principal las recetas que ha guardado. Si hace click en el icono del corazón eliminará esa receta de su lista.

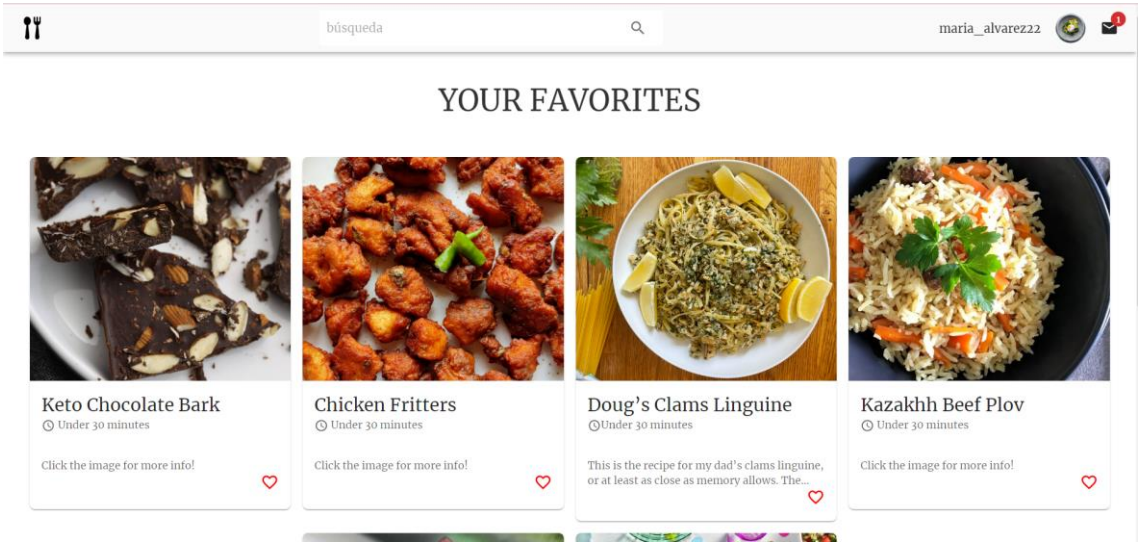


Figura A.14 Página de favoritos del usuario

A su vez, si no tiene ningún favorito marcado, se le indicará cómo puede guardar alguno.

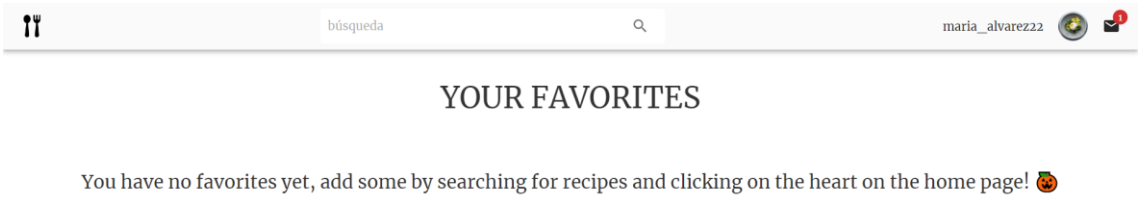


Figura A.15 Página de favoritos sin recetas

Con la barra de búsqueda puede introducir texto para obtener resultados relacionados con esa búsqueda.

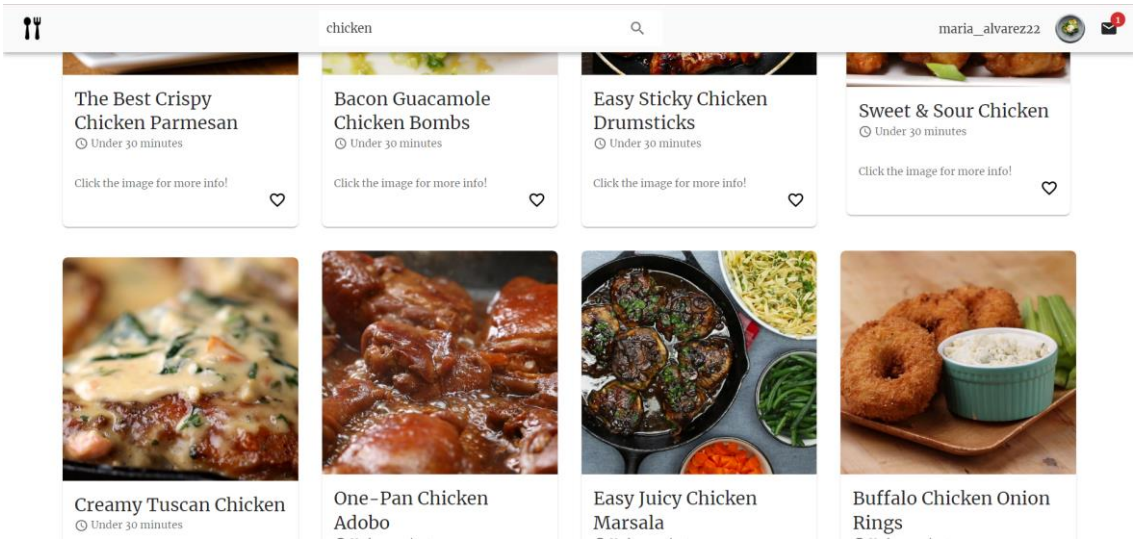


Figura A.16 Búsqueda de recetas con un ingrediente concreto

Con el enlace de cerrar sesión del menú del usuario será redirigido a la página de login, y con el logo de la aplicación de la figura A.17 puede volver en cualquier momento a la página principal.

Figura A.17 Logo de la aplicación

Las siguientes figuras se corresponden con vistas de la página web desde un dispositivo móvil.

Figura A.18 Vista desde móvil de la página principal

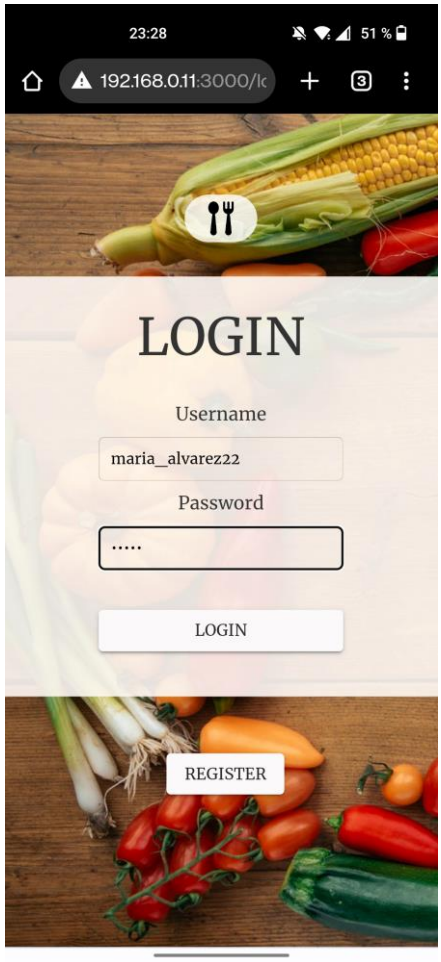


Figura A.19 Vista desde móvil de la página de login

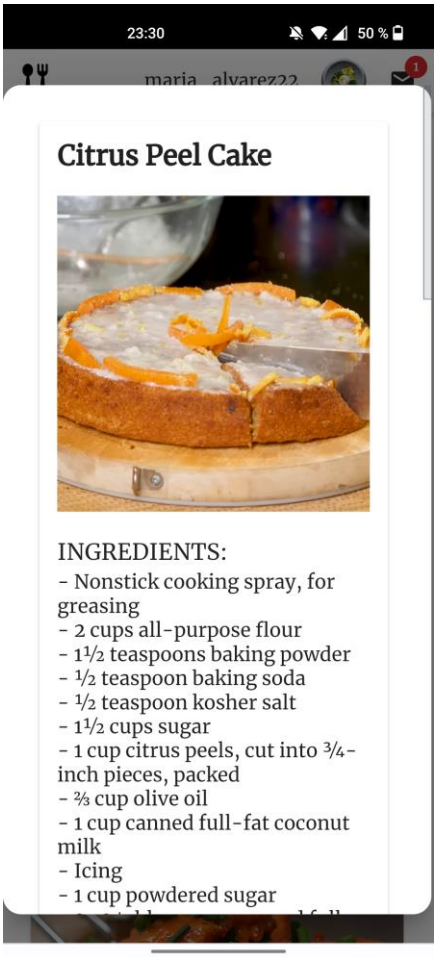


Figura A.20 Vista desde móvil de una receta abierta

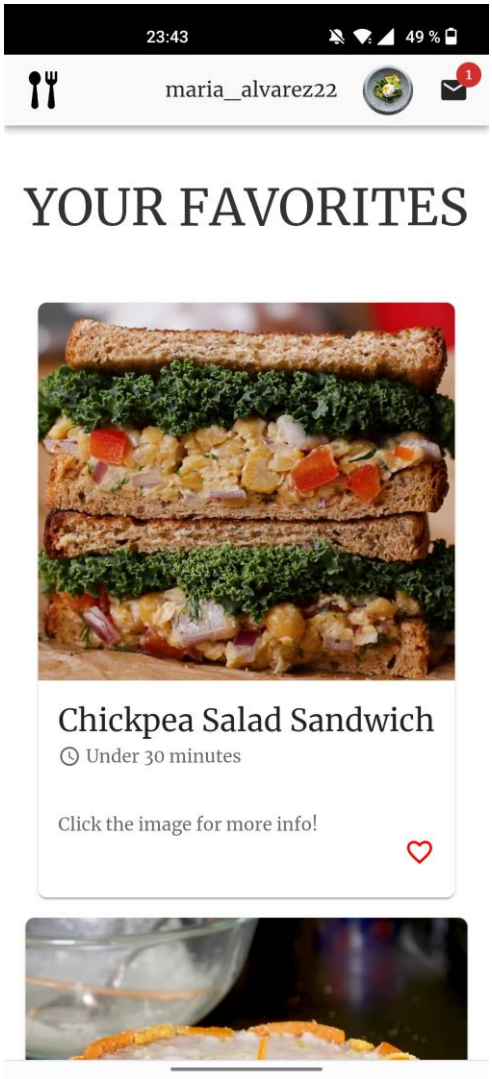


Figura A.21 Vista desde móvil de la página de favoritos

4 Evaluación

Una vez visualizado y explicado el producto resultante, se procede a evaluarlo para asegurar que cumple como solución para los problemas expuestos, y que se han llevado a cabo los requisitos y los objetivos iniciales.

Como ya se ha tratado a lo largo de esta memoria, el proyecto surge como solución a dos problemas detectados:

- La disponibilidad de una arquitectura que se encargue de todo el proceso de recopilación y almacenamiento de recetas culinarias.
- La visualización de recetas en una aplicación web que se adapte a las diferentes pantallas de los dispositivos, y que sea sencilla e intuitiva de usar.

Evaluando el sistema se han obtenido las siguientes conclusiones:

- La arquitectura del sistema cumple los objetivos de estar desarrollada en microservicios independientes, facilitando su futuro uso por otros desarrolladores para sus propios proyectos.
- El servidor de almacenamiento responde de manera ágil, y dispone de las 6000 recetas requeridas inicialmente.
- El servicio de acceso a los datos de MongoDB asegura su compatibilidad con otros sistemas, ya que las peticiones empleadas siguen el protocolo HTTP. Realiza todas las peticiones necesarias por la página web Zucca, incluyendo gestión de la sesión del usuario.
- La página web es completamente *responsive* para que los usuarios de Internet puedan acceder desde diferentes dispositivos, siendo lo más común el uso de los dispositivos móviles. La página cuenta con imágenes para captar la atención del público, y con un diseño sencillo para que su uso no requiera conocimientos previos.

En conclusión, los sistemas propuestos cumplen con los objetivos marcados, quedando ciertos aspectos pendientes de mejora en futuras versiones.

Conclusión

Para finalizar, se tratarán las aportaciones realizadas, las futuras mejoras del proyecto, los problemas encontrados durante la realización del mismo, y las opiniones personales obtenidas.

Aportaciones realizadas

Se ha llevado a cabo un correcto desempeño de la arquitectura del sistema de datos y de la página web de recetas, cumpliendo con los objetivos establecidos al inicio del proyecto.

Se ha planificado el proyecto en iteraciones para asegurar el cumplimiento de los plazos y evitar posibles retrasos. Posteriormente, se ha comenzado con el desarrollo de los servicios Docker, Kafka, back-end y front-end, realizando los despliegues de cada uno para poder pasar a la siguiente fase de desarrollo.

Con el servicio Docker se ha aportado un cluster de contenedores con las imágenes de MongoDB, Kafka y Zookeeper.

Con los servicios Kafka se dispone del MongoDB con recetas y categorías almacenadas, con una amplia variedad de datos a los que sacarle provecho cuando se implemente un sistema de visualización.

El back-end proporciona un servidor Node que implementa la comunicación y peticiones a la base datos, de forma que se puedan obtener usuarios, recetas y categorías.

Y, por último, el front-end permite la visualización e interacción de la información en una página web sencilla y funcional.

Trabajos futuros

Debido a la estructura de microservicios de la arquitectura, la implementación de mejoras en cada módulo se puede realizar de manera sencilla. A continuación, se proponen algunas de las mejoras pendientes para futuras versiones del sistema:

- **Sistema completo:**

  - **Subida a un host público**. En esta versión, la arquitectura está en local. Se propone subir los servidores a la nube de forma que estén siempre operativos para su uso.

- **Sistema de datos:**
  - **Automatización.** Actualmente, el sistema de recopilación y almacenamiento de datos tiene que ser ejecutado por el desarrollador paso a paso. Se propone de cara a futuro automatizar el despliegue del Docker Compose, el Kafka Producer y el Kafka Consumer con Jenkins, evitando tener que introducir manualmente los comandos.
  - **Archivos de configuración para contraseñas.** En algunos servicios como la conexión en el back-end con la base de datos, la contraseña se ve en claro en el fichero. Puesto que se trata de un proyecto local no supone ninguna brecha de seguridad, pero de cara a la subida del sistema para su uso público se propone definir estos parámetros en archivos ocultos.
- **Página web:**
  - **Nueva funcionalidad en la página web**. En este proyecto, al tratarse de una demostración de un uso de la arquitectura de datos, y, sumado al tiempo limitado de desarrollo, la página web Zucca tiene funcionalidad muy limitada. Se propone usar otros campos que tienen las recetas para añadir funcionalidad como filtrar los resultados por calorías, alergias, país… Así como poder cambiarse la foto de perfil o ver notificaciones.
  - **Cambio de idioma de la página web.** La página web de este proyecto muestra los datos recopilados de *tasty*, una API cuyo contenido está en inglés. Se propone la posibilidad de poder cambiar de idioma, traduciendo el texto al idioma deseado.
  - **Validación de datos en la página web.** Actualmente, el correo electrónico del usuario no se valida. Se propone realizar comprobaciones para que sigan la sintaxis de un correo electrónico válido.

Problemas encontrados

Los problemas más importantes surgidos mientras se implementaba el proyecto son los siguientes:

- Configuración de los servicios Kafka. Debido al desconocimiento de esta tecnología los problemas surgidos durante el desarrollo implicaron gran parte del tiempo. Algunos de ellos fueron que se excedía el tamaño máximo de mensaje, problemas con los puertos y que el Consumer no detectaba las nuevas publicaciones. Con la ayuda de búsquedas en Internet se fueron solucionando poco a poco.
- Mientras se realizaba el desarrollo de los microservicios de Kafka se hacían peticiones a la API *tasty* para ir probando su funcionamiento. Se debía tener cuidado con el número de peticiones que se hacían ya que el número de ellas al mes está limitado, lo que ralentizaba levemente el proceso.
- A la hora de desarrollar el front-end, se empezó el proyecto en Angular.js. Cuando ya se contaba con gran parte de la página principal desarrollada y se comenzó a implementar otros componentes y páginas, se consideró que la dificultad de desarrollo estaba aumentando demasiado, provocando mucho tiempo invertido intentando solucionar errores que no se terminaban de entender. Finalmente, se optó por utilizar la tecnología React.js, la cual destaca por tener una curva de aprendizaje más leve.
- El último de los problemas fue causado por un campo "_id" que algunas de las recetas de las categorías traían insertados de la API *tasty*. Puesto que solo ocurría en algunas ocasiones, provocaba errores inesperados en situaciones muy diferentes. Este problema se fue arrastrando hasta el final del desarrollo, ya que costó mucho tiempo encontrar la causa y en ocasiones parecía que estaba solucionado. Ese par clave valor solo debería existir cuando MongoDB lo crea al insertar el objeto en la base de datos, siendo de tipo "ObjectId". En cambio, el traído por la API era de tipo "Int32". Se solucionó eliminando ese campo innecesario, y permitiendo a Mongo autogenerar el suyo propio.

Opiniones personales

Desarrollar este sistema de manera prácticamente individual con tecnologías que nunca había utilizado ha sido un gran reto al que nunca me había enfrentado. Aunque han sido muchas horas dedicadas, y, unas cuantas, de frustración ante problemas inesperados y arrepentimiento de sumergirme en algo tan desconocido para mí, otras muchas han sido de satisfacción al ir viendo resultados en todo el esfuerzo empeñado.

Me ha encantado tratar todos los aspectos desde la recopilación de datos hasta el desarrollo de una aplicación web, y conocer nuevas tecnologías como lo han sido para mí React, MongoDB, Kafka y Node. Me llevo muchos conocimientos nuevos que me servirán en el futuro.

A pesar de esto, me quedo con las ganas de mejorar el sistema, ya que ha habido muchas ideas que no he tenido tiempo de implementar. Es por ello por lo que seguiré trabajando en él hasta estar completamente satisfecha, como reto personal.

Lista de referencias

[1] "API: qué es y para qué sirve." https://www.xataka.com/basics/api-que-sirve (accessed Jun. 25, 2022).

[2] RedHat, "¿Qué es una API de REST?" https://www.redhat.com/es/topics/api/what-is-a-rest-api (accessed Jun. 25, 2022).

[3] "Comunicando microservicios con Apache Kafka - Paradigma." https://www.paradigmadigital.com/dev/comunicacion-microservicios-apache-kafka/ (accessed Jun. 27, 2022).

[4] "Clúster y componentes de Apache Kafka - Documentación de IBM." https://www.ibm.com/docs/es/oala/1.3.5?topic=SSPFMY_1.3.5/com.ibm.scala.doc/config/iwa_cnf_scldc_apche_con_c.html (accessed Jun. 27, 2022).

[5] "Overview of Docker Compose | Docker Documentation." https://docs.docker.com/compose/ (accessed Jun. 25, 2022).

[6] "What Is a Framework?" https://www.codecademy.com/resources/blog/what-is-a-framework/ (accessed Jun. 27, 2022).

[7] "What is a Content Hub? (Examples, Types, SEO Benefits)." https://terakeet.com/blog/content-hub/ (accessed Jun. 27, 2022).

[8] “Salarios | Indeed.” https://es.indeed.com/career/salaries (accessed Jun. 25, 2022).

[9] “Kafka Consumers | Learn Apache Kafka with Conduktor.” https://www.conduktor.io/kafka/kafka-consumers (accessed Jun. 25, 2022).

[10] “Kafka Producers | Learn Apache Kafka with Conduktor.” https://www.conduktor.io/kafka/kafka-producers (accessed Jun. 25, 2022).

[11] Víctor Manuel Valle, “Comunicando microservicios con Apache Kafka - Paradigma.” https://www.paradigmadigital.com/dev/comunicacion-microservicios-apache-kafka/ (accessed Jun. 25, 2022).

[12] “¿Qué son y para qué sirven los microservicios?” https://www.redhat.com/es/topics/microservices (accessed Jun. 25, 2022).

[13] “MODELO DE PUBLICACIÓN/SUSCRIPCIÓN - Análisis de rendimiento de protocolos de publicación/subscr.” https://1library.co/article/modelo-publicaci%C3%B3n-suscripci%C3%B3n-an%C3%A1lisis-rendimiento-protocolos-publicaci%C3%B3n-subscr.qvrn56ry (accessed Jun. 25, 2022).

[14] “Pruebas alfa - Globe Testing.” https://ahorasomos.izertis.com/globetesting/pruebas-alfa/ (accessed Jun. 25, 2022).

[15] “Diseño Web Responsive — Cómo hacer que un sitio Web se vea bien en Teléfonos y Tabletas.” https://www.freecodecamp.org/espanol/news/diseno-web-responsive-como-hacer-que-un-sitio-web-se-vea-bien-en-telefonos-y-tabletas/ (accessed Jul. 05, 2022).

[16] R. Cañadas, “Base de datos relacional | Qué es, Características y Ejemplos.” https://abdatum.com/business-intelligence/base-datos-relacional (accessed Jun. 25, 2022).

[17] R. Cañadas, “Base de datos no relacional o NoSQL | ¿Cómo funcionan?” https://abdatum.com/business-intelligence/base-datos-no-relacional (accessed Jun. 25, 2022).

[18] “Apache Kafka vs Amazon Kinesis - Comparación de configuración, rendimiento, seguridad y precio | solucionador.” https://www.upsolver.com/blog/comparing-apache-kafka-amazon-kinesis (accessed Jun. 27, 2022).

[19] “What Is The MEAN Stack? Introduction & Examples | MongoDB.” https://www.mongodb.com/mean-stack (accessed Jun. 25, 2022).

[20] “What Is The MERN Stack? Introduction & Examples | MongoDB.” https://www.mongodb.com/mern-stack (accessed Jun. 25, 2022).

[21] “The web framework for perfectionists with deadlines | Django.” https://www.djangoproject.com/ (accessed Jun. 25, 2022).

[22] “Kaggle y otras plataformas alternativas para aprender ciencia de datos | datos.gob.es.” https://datos.gob.es/es/blog/kaggle-y-otras-plataformas-alternativas-para-aprender-ciencia-de-datos (accessed Jun. 25, 2022).

[23] “API Hub - Free Public & Open Rest APIs | RapidAPI.” https://rapidapi.com/hub (accessed Jun. 25, 2022).

[24] “Guía básica de Maven.” https://javadesde0.com/guia-basica-maven/ (accessed Jun. 27, 2022).

[25] “Kafka Broker, Kafka Topic, Consumer and Record Flow in Kafka | by Kajal Rawal | Medium.” https://kajalrawal.medium.com/kafka-broker-kafka-topic-consumer-and-record-flow-in-kafka-ec55104977b8 (accessed Jun. 27, 2022).

[26] “Tasty API Documentation (apidojo) | RapidAPI.” https://rapidapi.com/apidojo/api/tasty (accessed Jun. 27, 2022).

[27] A. Mousinho, “SEO: guía completa del posicionamiento en buscadores [2021],” Jun. 03, 2020. https://rockcontent.com/es/blog/que-es-seo/ (accessed Jun. 25, 2022).

[28] X. Vilajosana, G. Leandro, and N. Moldes, “Arquitectura de aplicaciones web”.

[29] “ReactDOM – React.” https://es.reactjs.org/docs/react-dom.html (accessed Jun. 26, 2022).

[30] “package.json | npm Docs.” https://docs.npmjs.com/cli/v7/configuring-npm/package-json (accessed Jun. 29, 2022).

[31] “Persistencia de la información en docker - PLEDIN 3.0.” https://plataforma.josedomingo.org/pledin/cursos/openshift/curso/u05/ (accessed Jun. 28, 2022).

[32] “How to Run a Python Script - GeeksforGeeks.” https://www.geeksforgeeks.org/how-to-run-a-python-script/ (accessed Jun. 28, 2022).

[33] “Using Requirement Files – Real Python.” https://realpython.com/lessons/using-requirement-files/ (accessed Jun. 28, 2022).

[34] “BSON - Eficiencia, Tipos de datos y sintaxis | KripKit.” https://kripkit.com/bson/ (accessed Jun. 29, 2022).

[35] “Librería Axios: cliente HTTP para Javascript.” https://desarrolloweb.com/articulos/axios-ajax-cliente-http-javascript.html (accessed Jun. 30, 2022).

[36] “Usando el Hook de efecto – React.” https://es.reactjs.org/docs/hooks-effect.html (accessed Jun. 29, 2022).

[37] “Context – React.” https://es.reactjs.org/docs/context.html (accessed Jun. 30, 2022).

[38] “Usando el Hook de estado – React.” https://es.reactjs.org/docs/hooks-state.html (accessed Jun. 29, 2022).

[39] “Connection String URI Format — MongoDB Manual.” https://www.mongodb.com/docs/manual/reference/connection-string/ (accessed Jun. 29, 2022).

[40] “REST API Design Best Practices for Parameter and Query String Usage | Moesif Blog.” https://www.moesif.com/blog/technical/api-design/REST-API-Design-Best-Practices-for-Parameters-and-Query-String-Usage/ (accessed Jun. 29, 2022).

[41] “kafka-python — documentación de kafka-python 2.0.2-dev.” https://kafka-python.readthedocs.io/en/master/ (accessed Jul. 01, 2022).

[42] “snappy | A fast compressor/decompressor.” http://google.github.io/snappy/ (accessed Jul. 01, 2022).

[43] “PyMongo 4.1.1 Documentation — PyMongo 4.1.1 documentation.” https://pymongo.readthedocs.io/en/stable/ (accessed Jul. 01, 2022).

Anexo A: Manual de instalación

Este manual de instalación explica paso a paso cómo desplegar la arquitectura de manera local en un sistema operativo Windows. Cabe mencionar que en algunos apartados puede haber diversas formas de obtener los mismos resultados. Se recomienda seguir lo especificado en este manual para asegurar el correcto despliegue y funcionamiento del sistema.

Requerimientos

- Docker desktop
- Java 11
- Entorno de Python versión “3.7”
- Node versión 16.14.0
- Git versión “2.24.1.windows.2” o superiores

1. Instalación de las tecnologías necesarias.
    a. Descargar Docker Desktop desde la página oficial.
    b. Descargar Java SE Development Kit 11.0.15 desde la página oficial.
    c. Descargar Python 3.7 desde la página oficial.
    d. Descargar Node desde la página oficial.
    e. Descargar Git desde la página oficial.
2. Descargar del repositorio de GitHub “Zucca” los proyectos de los cuatro microservicios.

Figura A.1 Proyectos de cada servicio en GitHub

3. Abrir el proyecto ***docker-services*** y ejecutar el archivo "*docker-compose.yml*" con el comando siguiente:

```
PS C:\Users\Maria\visual-workspace\Zucca\docker-services>
docker-compose up
```

Figura A.2 Comando para levantar Docker Compose

4. Abrir Docker Desktop para asegurarse de que se han desplegado correctamente los contenedores.

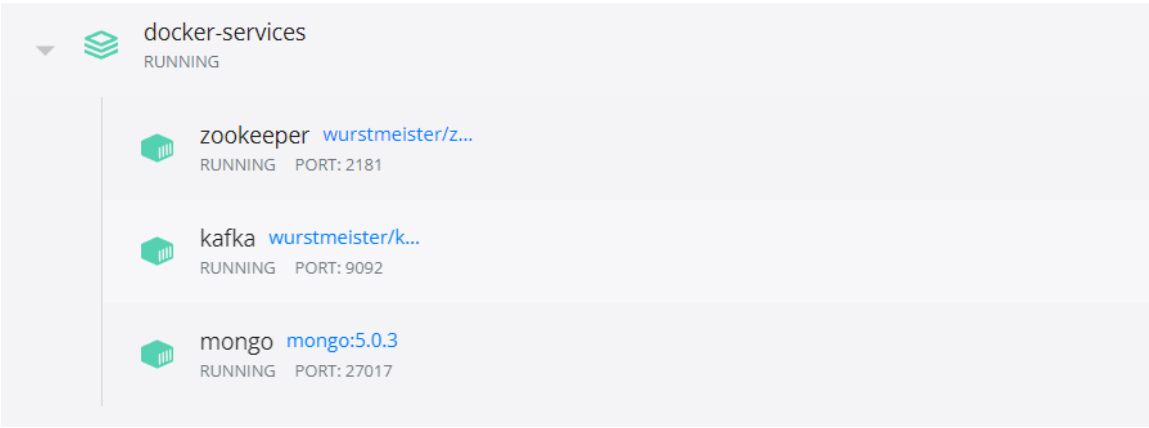


Figura A.3 Contenedores Kafka, ZooKeeper y MongoDB en Docker Desktop

5. Abrir el proyecto **kafka-producer** con un entorno apto para proyectos Java y Maven. Se recomienda IntelliJ IDEA para poder seguir los pasos de este manual.
6. Configurar el proyecto pulsando "CTRL + ALT + SHIFT + S". Se abrirá la ventana de la figura 4.

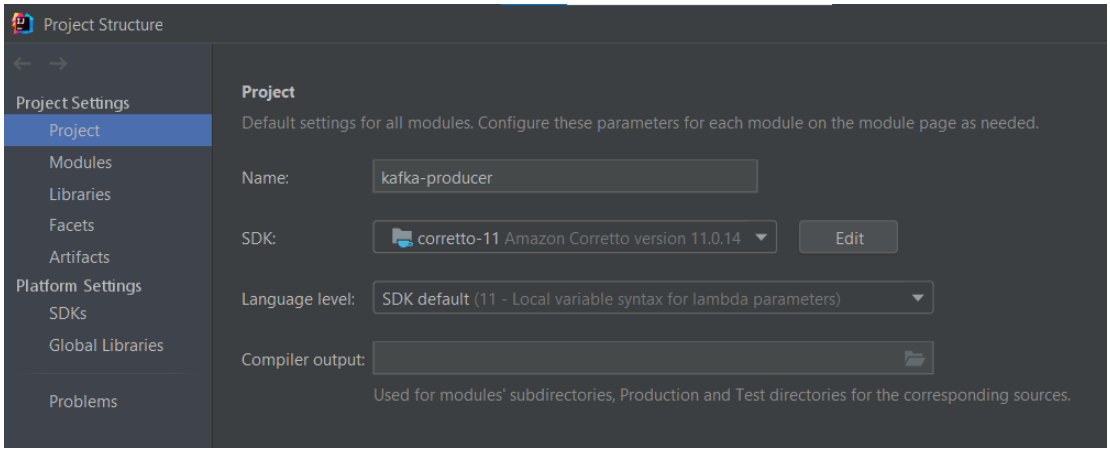


Figura A.1 Ajustes del compilador del proyecto Java

Aquí, en el apartado “Project”, se seleccionará la versión de Java 11 previamente instalada. Si no se dispone de ella, se puede instalar desde el propio IDE.

7. Abrir el proyecto **kafka-consumer** desde un entorno apto para Python. Se recomienda el uso de PyCharm.
8. En la esquina inferior derecha, hay una pestaña para configurar el entorno. Se observa en la figura 5.

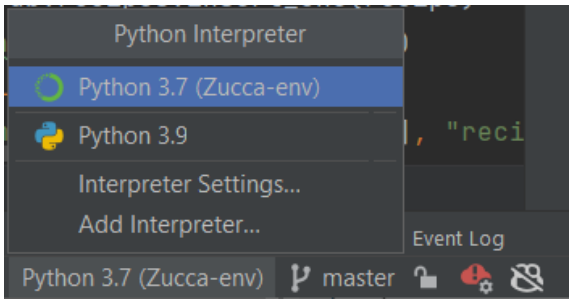


Figura A.2 Entorno de Python desde PyCharm

Desde aquí, se puede crear o configurar el entorno. Si se hace click en “Add Interpreter…” se abrirá la siguiente ventana para crear un entorno

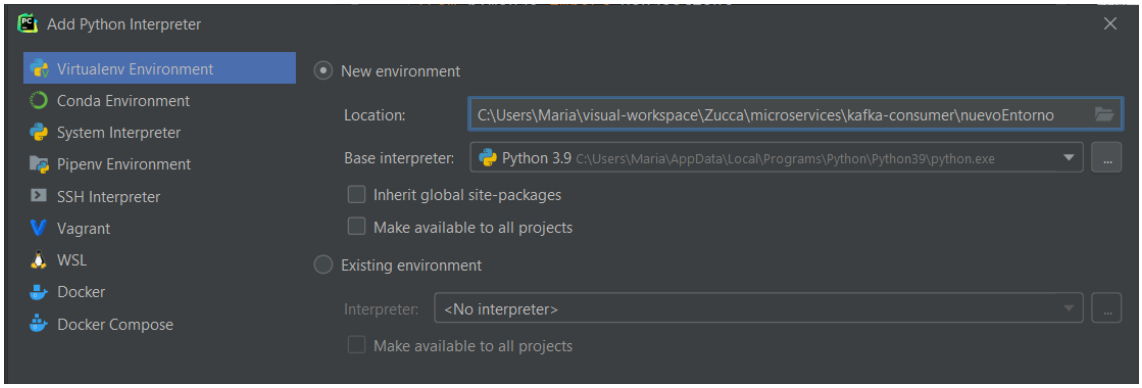


Figura A.3 Crear un entorno Python desde PyCharm

Se debe seleccionar la localización donde se desea crear el entorno y el ejecutable de Python que instalamos en el paso 1.

* El crear un entorno para cada proyecto no es requisito obligatorio, pero no hacerlo puede provocar conflictos entre paquetes de diferentes proyectos.

9. Asegurarse de que está seleccionado el entorno recién creado. Se debería visualizar como en la figura 5.
10. En una terminal en el directorio “kafka-consumer\resources” se deben instalar las dependencias ejecutando el comando de la figura 7.

```
PS C:\Users\Maria\visual-workspace\Zucca\microservices\kafka-consumer\resources>
pip install .\requirements.txt
```

Figura A.4 Comando para instalar dependencias Kafka Consumer

11. Abrir el proyecto el **back-end** con, por ejemplo, Visual Studio Code.
12. Realizar el siguiente comando para instalar todas las dependencias incluidas en el “package.json”.

```
PS C:\Users\Maria\visual-workspace\Zucca\back-end>
pip install
```

Figura A.5 Comando para instalar dependencias Backend

13. Abrir el proyecto el **front-end** con, por ejemplo, Visual Studio Code.
14. Realizar el siguiente comando para instalar todas las dependencias incluidas en el “package.json”, es el mismo que para back-end (ambos proyectos están programados en Javascript).

```
PS C:\Users\Maria\visual-workspace\Zucca\front-end>
pip install
```

Figura A.6 Comando para instalar dependencias Frontend

Una vez tengamos los proyectos configurados en local, se procede a explicar su despliegue.

1. En la herramienta Docker Desktop, darle click al botón con el símbolo de “play” para arrancar los contenedores.

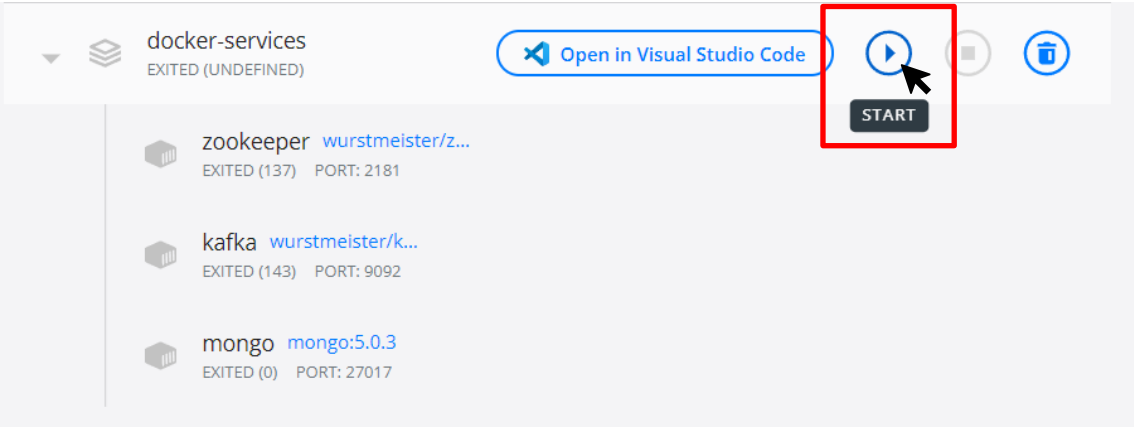


Figura A.7 Arrancar contenedores Docker

Si se despliegan correctamente, se verán como la figura 11.

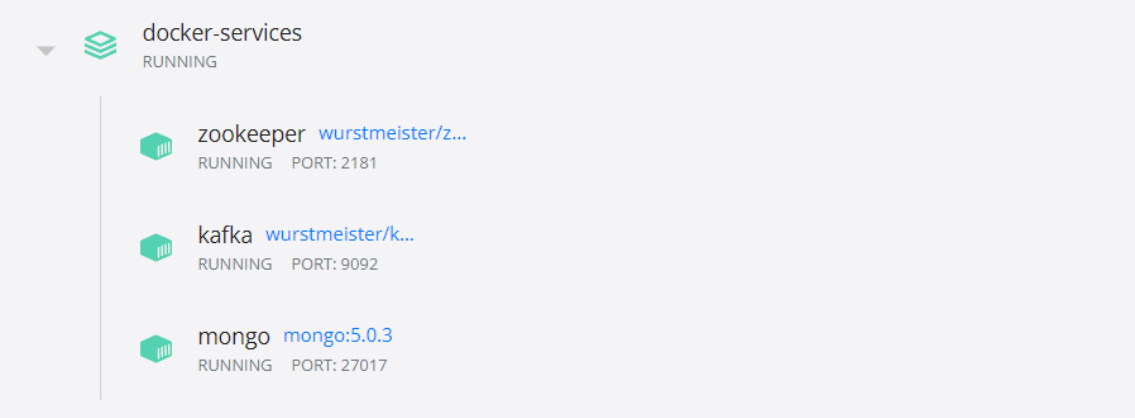


Figura A.8 Contenedores Docker desplegados

2. En el proyecto kafka-consumer, ejecutar el script de Python. Se puede hacer desde PyCharm dándole al botón de ejecutar de la figura 10.

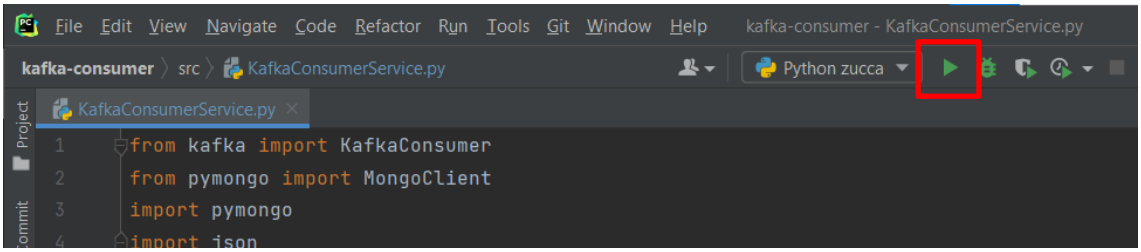


Figura A.9 Ejecución Kafka Consumer

Si se da el caso de que no detecta automáticamente el fichero que debe ejecutar, hacer click en la pestaña de la figura 11.

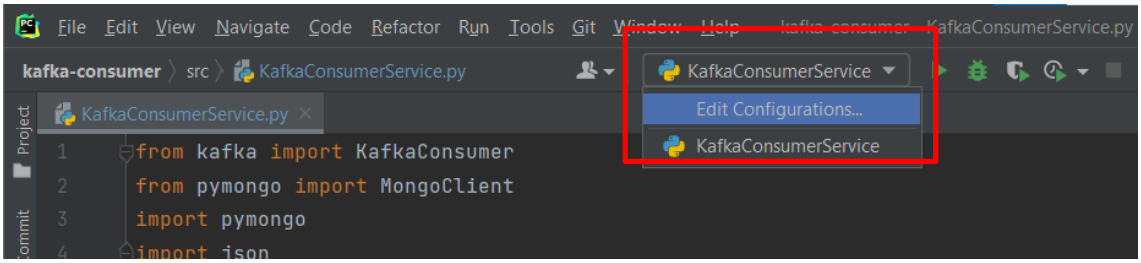


Figura A.10 Fichero a ejecutar en el Kafka Consumer

Al pinchar en “Edit Configurations…”, se abrirá la siguiente ventana, donde se debe añadir una configuración de ejecución. En “Script Path” se debe seleccionar el script “KafkaConsumerService.py”.

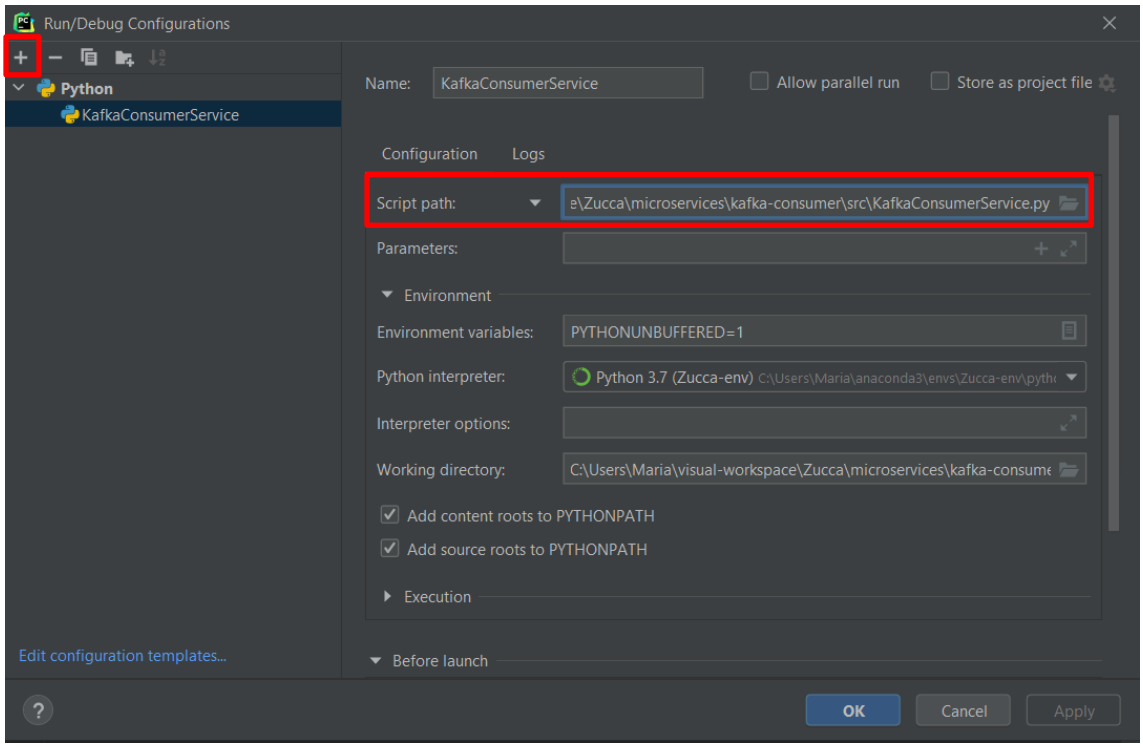


Figura A.11 Configuración de ejecución Kafka Consumer

Probar de nuevo a darle al botón de ejecutar.

3. En el proyecto kafka-producer, ejecutar la clase “KafkaProducerApplication.java”, ya que es la que contiene el método “main”.

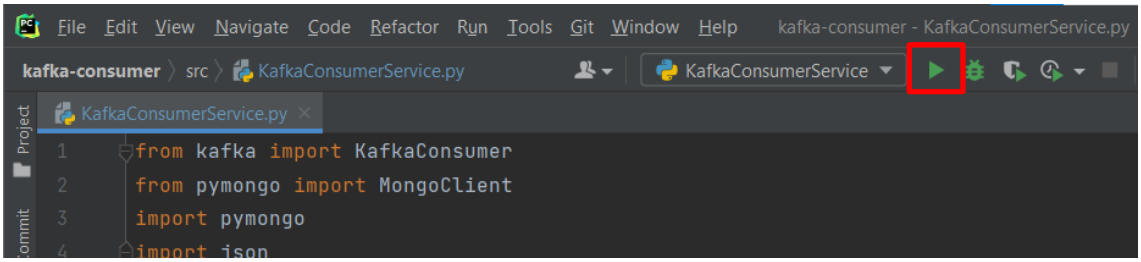


Figura A.12 Ejecución del Kafka Producer

Si se da el caso de que no detecta automáticamente la clase que debe ejecutar, hacer click en la pestaña de la figura 14.

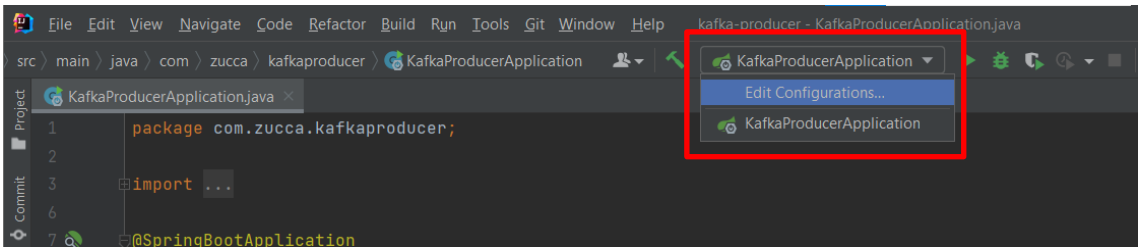


Figura A.13 Fichero a ejecutar en el Kafka Producer

Al pinchar en “Edit Configurations…”, se abrirá la ventana de la figura 15, donde se debe añadir una configuración de ejecución. En el campo del recuadro rojo rectangular, se debe seleccionar la clase “KafkaProducerApplication”.

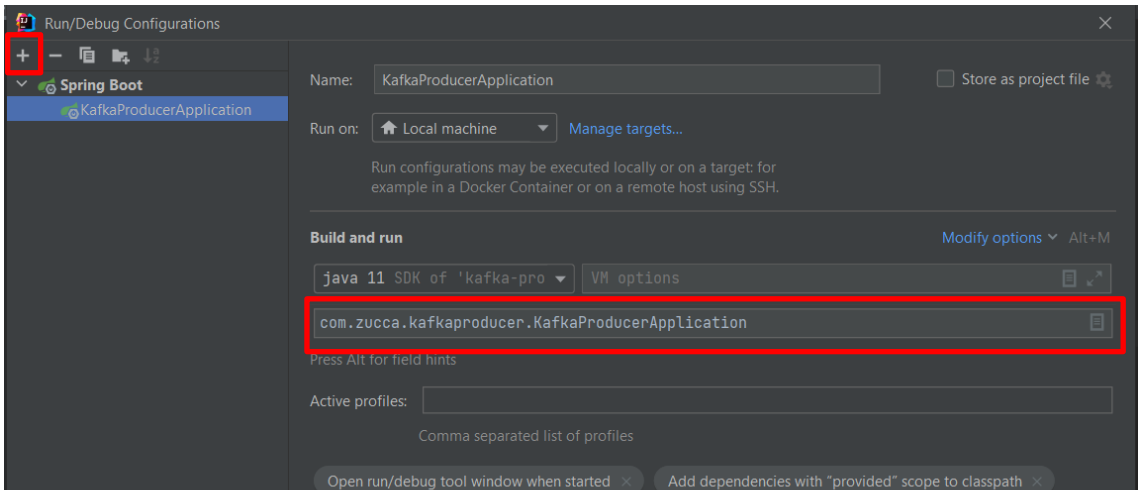


Figura A.14 Configuración de ejecución Kafka Producer

Probar de nuevo a darle al botón de ejecutar.

4. En la consola del Kafka Producer se puede visualizar el número de peticiones que lleva realizadas. Se puede parar la ejecución cuando se desee. En este proyecto se para al realizar las 150 vueltas del bucle que realiza las peticiones.

5. En el proyecto del back-end, ejecutar el siguiente comando para desplegar el servidor en el puerto 5000.

```
PS C:\Users\Maria\visual-workspace\Zucca\back-end>
npm run dev
```

Figura A.15 Comando para ejecutar back-end

6. En el proyecto del front-end, ejecutar el comando siguiente:

```
PS C:\Users\Maria\visual-workspace\Zucca\front-end>
npm start
```

Figura A.16 Comando para ejecutar front-end

7. Si todo ha ido correctamente, se puede abrir un navegador y acceder a la dirección “localhost:3000”, donde estará desplegada la página web.

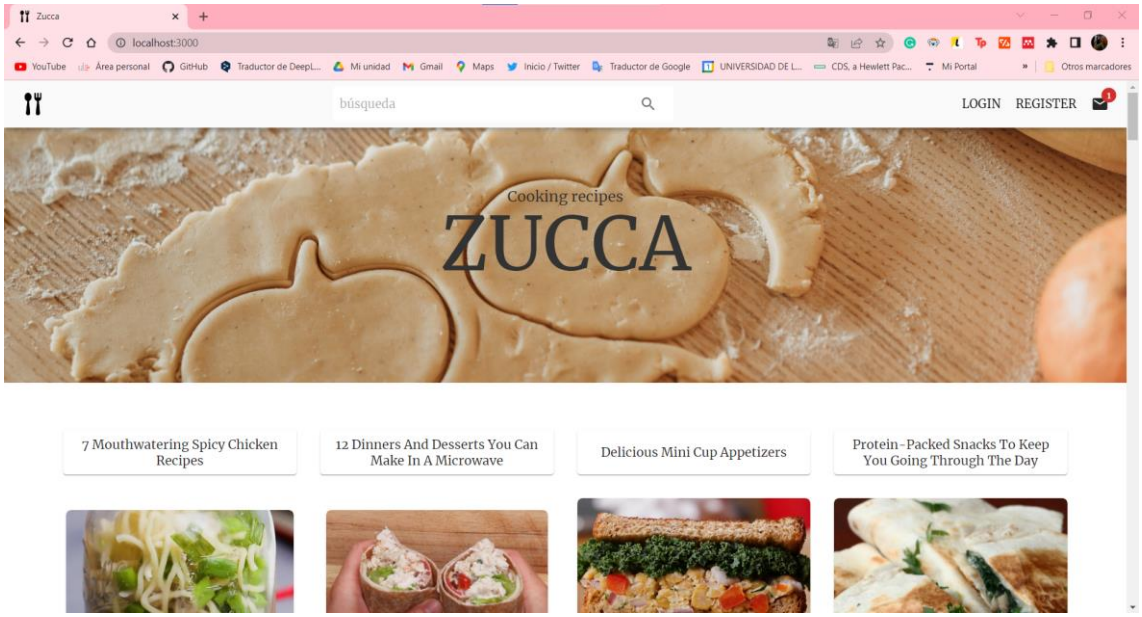


Figura A.17 Página web

Anexo B: Manual de usuario de la página web

Este manual de usuario explica detalladamente qué acciones se pueden llevar a cabo en la página web, y cómo. Para poder utilizar este servicio es necesario realizar los pasos especificados en el “Manual de instalación”, adjunto en el anexo A.

- **ACCEDER A LA WEB**

Al acceder a la página web en la dirección “localhost:3000” por primera vez, visualizará la página principal de la figura B.1.

En ella, hay una cabecera en la parte superior, una imagen que se corresponde con la marca, cuatro categorías aleatorias y las recetas, también aleatorias.

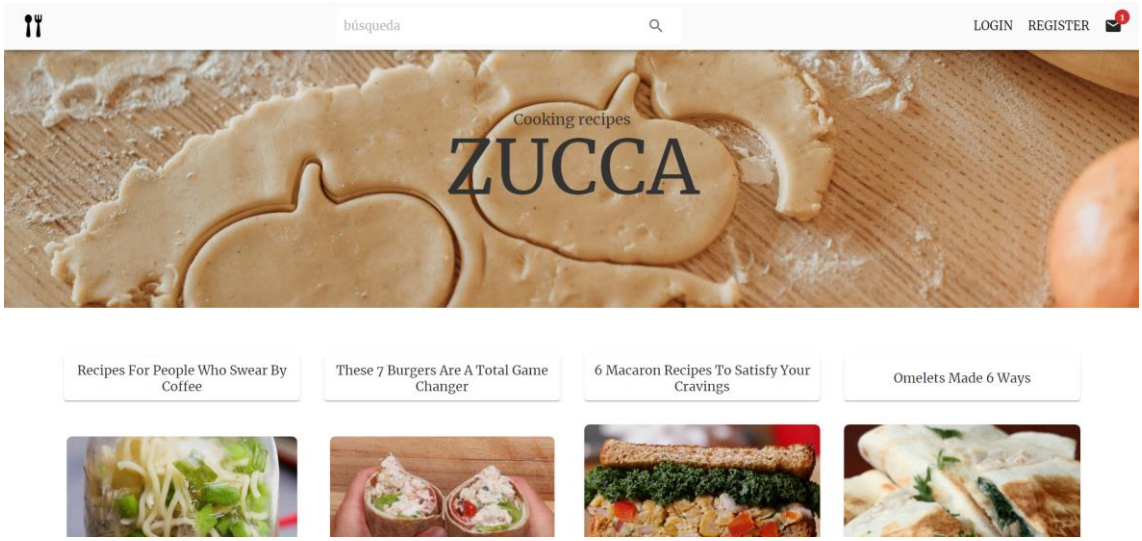


Figura B. 2 Página principal

- **VISUALIZAR RECETAS**

Para visualizar recetas, debe desplazarse hacia abajo por la página web. Por cada receta verá una imagen, su título, un tiempo aproximado de preparación y, en las que esté disponible, una descripción. También el icono para marcar una receta como favorita, lo que se detallará más adelante.

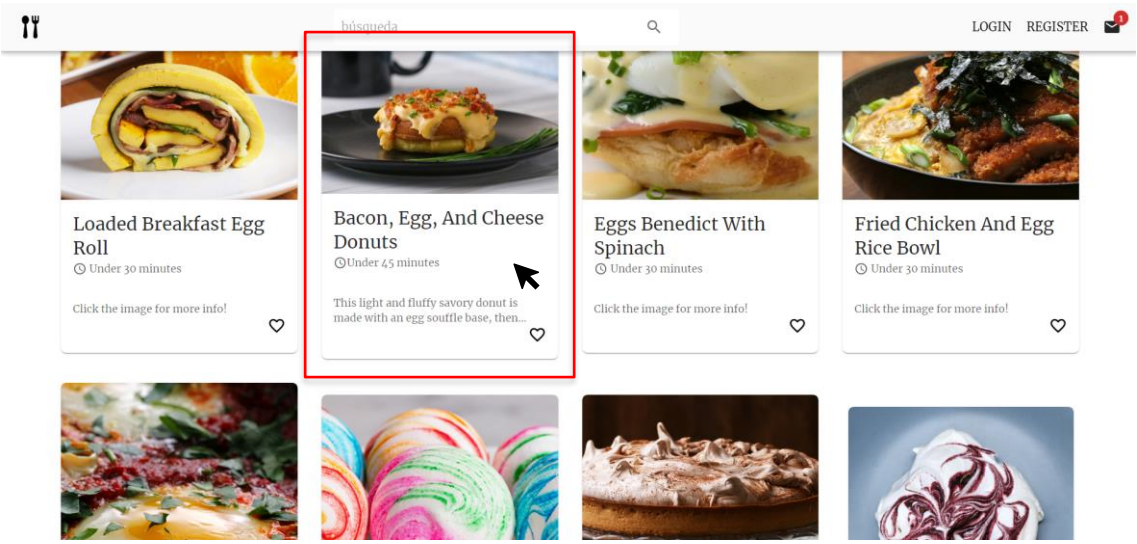


Figura B. 3 Recetas de la página principal

Para ver los pasos, medidas e ingredientes de una receta, debe hacer click en ella. Para cerrarla, se puede hacer click fuera de la ventana emergente, o pulsar la tecla “Esc” del teclado.

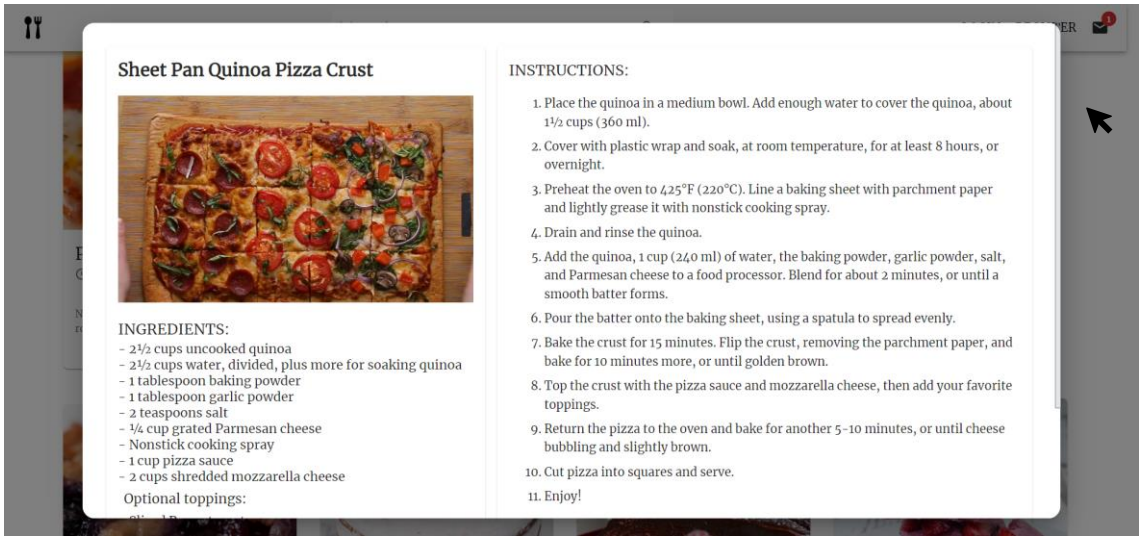


Figura B. 4 Receta detallada

- **EXPLORAR POR CATEGORÍAS**

En la parte superior de la página principal, cuenta con cuatro menús de categorías que también se cargan aleatoriamente al entrar. Si se hace click en alguna de ellas, se le mostrarán sus recetas correspondientes. Puede pinchar en la deseada para visualizar su información.

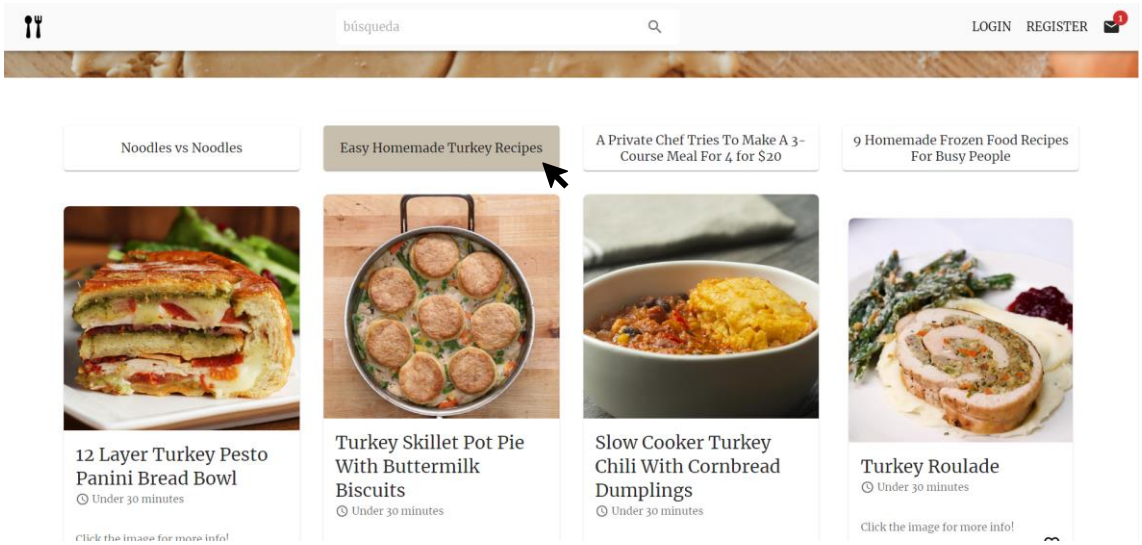


Figura B. 5 Categoría seleccionada

Para salir de la categoría y volver a ver todas las recetas, basta con volver a hacer click en su nombre.

- **BÚSQUEDA DE RECETAS**

Si desea buscar recetas cuyo título incluya una palabra determinada, debe hacer uso de la barra de búsqueda situada en el *header*, en la parte superior. En el

ejemplo de la figura B.5 se realiza una búsqueda de “omelet”, obteniendo todas las recetas relacionadas con “tortilla” (traducido al español).

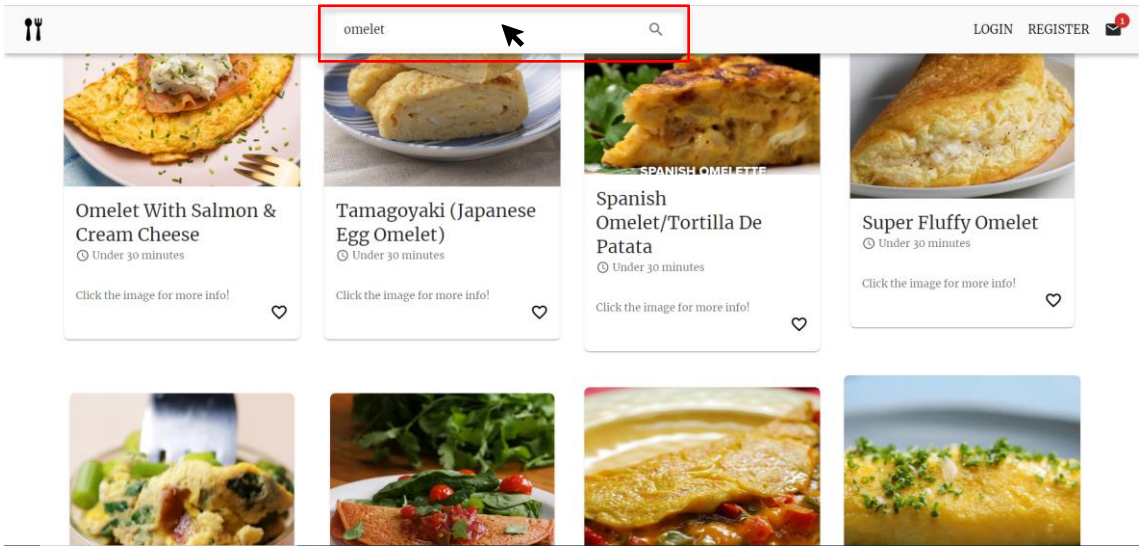
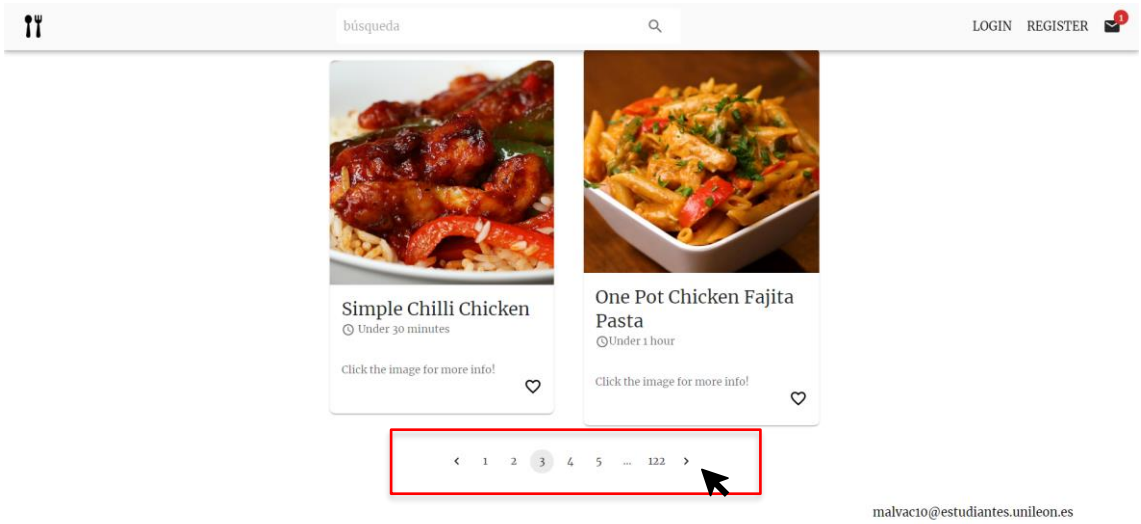


Figura B. 6 Búsqueda de recetas relacionadas con "omelet"

- **OBTENER MÁS RECETAS**

Si quiere visualizar más recetas, puede desplazarse al final de la página, donde hay un elemento destinado a moverse entre páginas. Haciendo uso de las flechas y los números, puede avanzar y retroceder entre ellas.



- **INICIAR SESIÓN**

Para iniciar sesión en la aplicación, debe hacer click en el texto “LOGIN”, en la parte derecha de la cabecera superior.

Figura B. 7 Enlace a la página de iniciar sesión

También, puede acceder desde la página de registrarse, donde habrá un botón con el texto “LOGIN”.

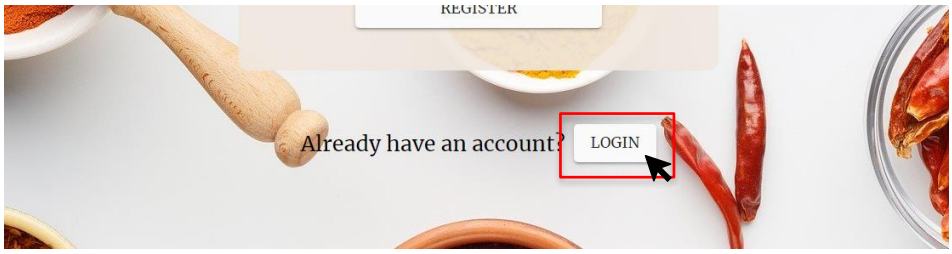


Figura B. 8 Botón de iniciar sesión desde la página de registrarse

Será redirigido a la página de iniciar sesión, donde tiene el formulario para introducir su nombre de usuario y contraseña. Si introduce unas credenciales incorrectas, se le mostrará una alerta avisándole.

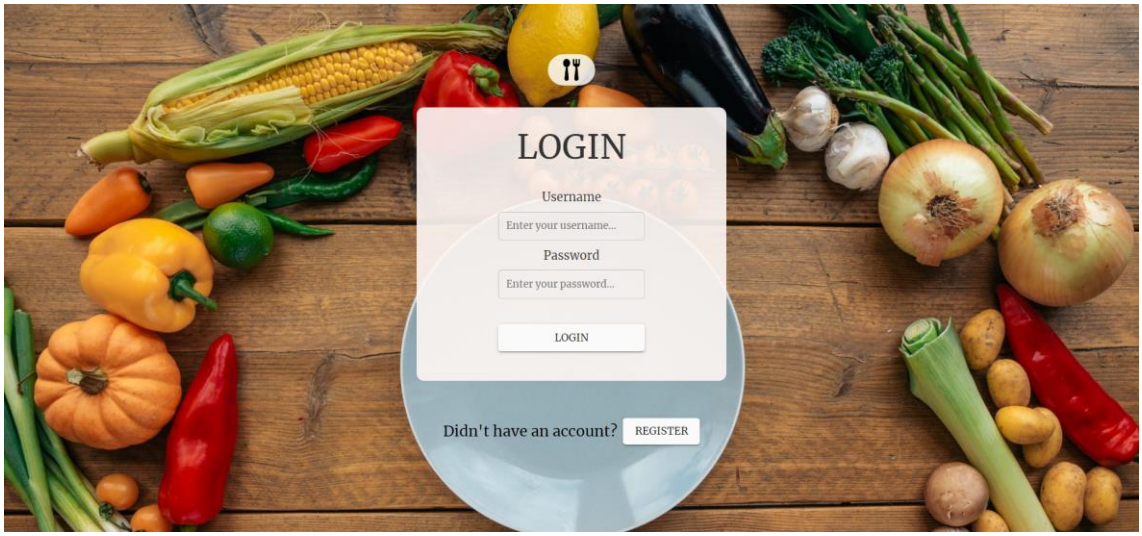


Figura B. 9 Página de iniciar sesión

- **REGISTRARSE**

Para registrarse en la aplicación, puede o bien acceder desde la página de iniciar sesión haciendo click en el botón de “REGISTER”, o desde la página principal, en el enlace “REGISTER”.

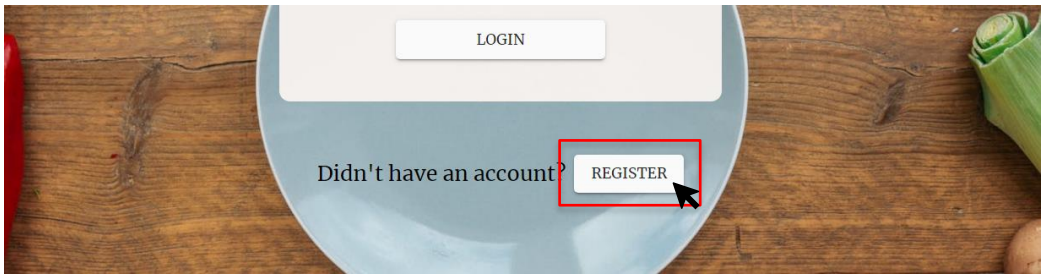


Figura B. 10 Botón de registrarse desde la página de iniciar sesión

Figura B. 11 Enlace a la página de registrarse

Será redirigido a la página de la figura B.11. En el formulario debe introducir un nombre de usuario, un email y una contraseña. El nombre de usuario no puede

coincidir con el de otros usuarios registrados, por lo que si no está disponible se le mostrará un aviso para que escoja otro.

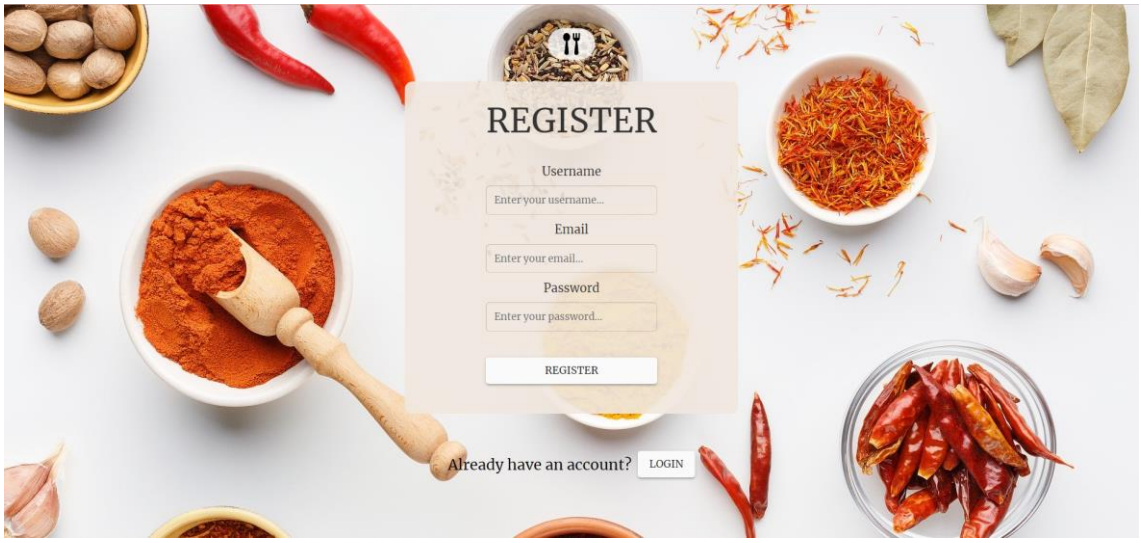


Figura B. 12 Página de registrarse

- **AÑADIR RECETA COMO FAVORITA**

Solo si ha iniciado sesión en la aplicación podrá guardar recetas para consultarlas más tarde. Al hacer click en el icono de corazón de una receta, este se marcará con el color rojo para indicar que ya está guardada. Puede volver a hacer click para desmarcarla y eliminarla de favoritos.

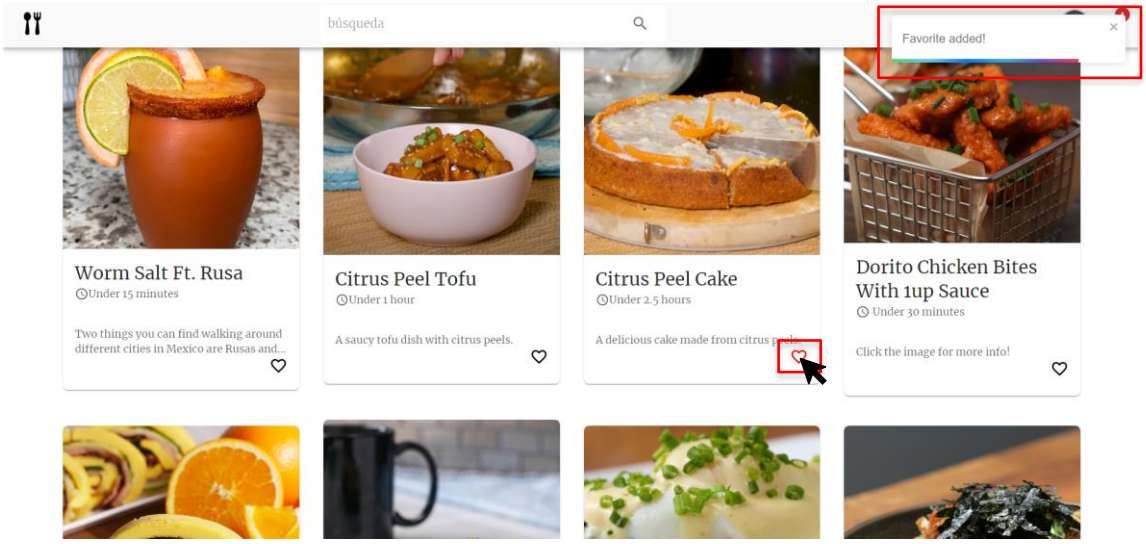


Figura B. 13 Receta favorita en la página principal

- **ELIMINAR RECETA DE FAVORITOS**

Para eliminar una receta de sus favoritos, siga el mismo proceso que para añadirla. Solo con hacer click en el icono del corazón, que debe estar en color rojo, se borrará de las recetas guardadas del usuario. Puede hacerlo tanto desde la página principal como desde la de favoritos.

- **VISUALIZAR FAVORITOS**

Para acceder a su página personal de recetas guardadas, debe hacer click en el icono de su avatar en la cabecera, al lado de su nombre usuario. En él, tiene el enlace a “Favorites”, como se ve en la figura B.13.

Figura B. 14 Menú para acceder a los favoritos

Será redirigido a la página de la figura B.14, donde visualizará todas sus recetas favoritas. Si hace click en el icono del corazón, se eliminará esa receta de sus favoritos.

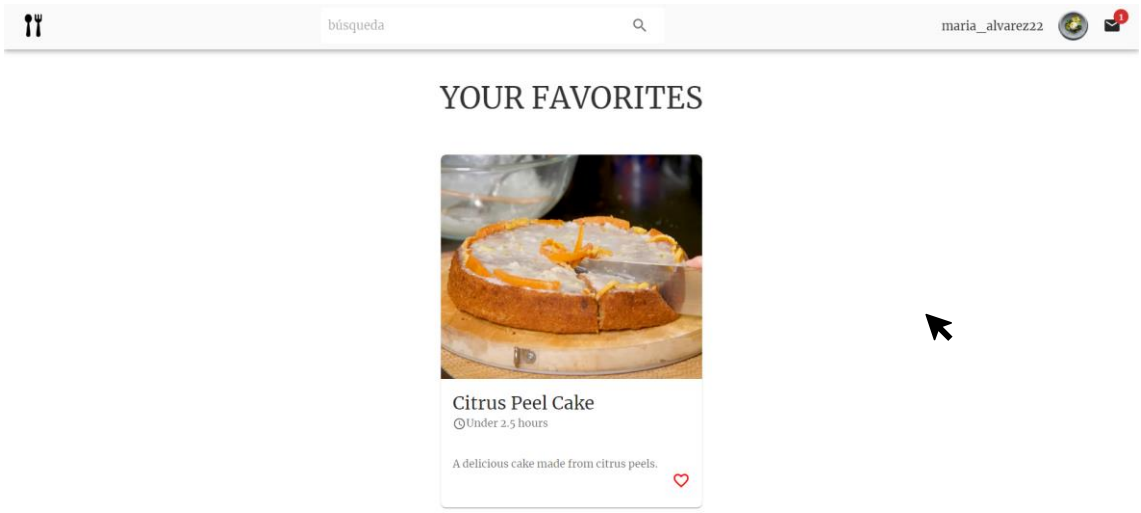


Figura B. 15 Página de favoritos del usuario

Si la página que se le muestra es la de la figura B.15, es que no tiene ninguna receta guardada. Como se le indica, puede ir a la página principal y realizar los pasos anteriores.

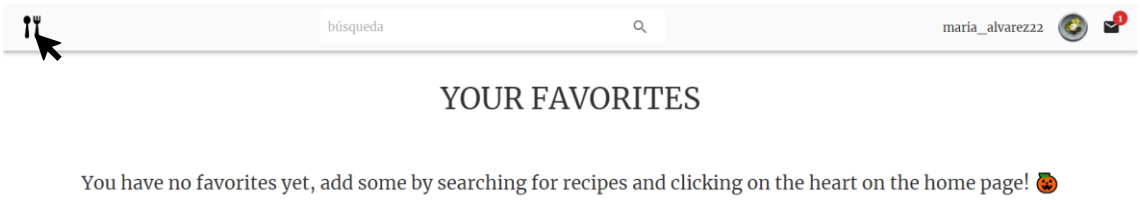


Figura B. 16 Página de favoritos vacía

- **CERRAR SESIÓN**

Para cerrar sesión debe hacer click en su avatar de usuario, y en “Logout”. Será redirigido de nuevo a la página de iniciar sesión por si desea entrar con otra cuenta, o crearse una nueva.

Figura B. 17 Menú para cerrar sesión

- **VOLVER A LA PÁGINA PRINCIPAL**

Tanto en la página de favoritos, como en la de iniciar sesión y registrarse, dispone del logo de la aplicación para volver a la página principal en cualquier momento. Solo debe hacer click en él. Este, se muestra en la figura B.17.

Figura B. 18 Logo de la aplicación

La sesión de usuario se mantiene cuando cierra el navegador, por lo que si lo vuelve a abrir seguirá logueado. Para cerrar sesión siga los pasos indicados en el apartado “Cerrar sesión”.