UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Alcoy

Desarrollo de una plataforma de recomendación de recetas de cocina basada en tecnologías de inteligencia artificial y computación en la nube

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Sillero Salvatierra, Kilian

Tutor/a: Esparza Peidro, Javier

CURSO ACADÉMICO: 2022/2023

Resumen

El presente trabajo se ha centrado en el diseño e implementación de una arquitectura 'serverless' para servir una aplicación web de creación de recetas basada en la Inteligencia Artificial. La aplicación web ha sido desarrollada en React y utiliza la API de ChatGPT para generar dichas recetas a partir de las instrucciones del usuario.

Para ello, se han utilizado diversos servicios de la plataforma de computación en la nube Amazon Web Services, como son: Lambda, API Gateway, DynamoDB, Cognito entre otros. Se ha llevado a cabo un proceso de análisis y diseño previo, identificando los requerimientos funcionales y no funcionales del sistema, y se ha seguido una metodología ágil para la gestión del proyecto. El resultado obtenido es una arquitectura eficiente y escalable, que permite la gestión y almacenamiento de datos de manera segura y el procesamiento de las solicitudes del usuario de forma rápida y eficaz.

Palabras clave: Serverless; AWS; GPT, React.

Abstract

This work has focused on the design and implementation of a 'serverless' architecture to serve an AI-based recipe creation web application. The web application has been developed using React and utilizes the ChatGPT API to generate recipes based on user instructions.

To achieve this, various services from the Amazon Web Services cloud computing platform have been used, such as Lambda, API Gateway, DynamoDB, Cognito, among others. A prior analysis and design process was carried out, identifying the functional and non-functional requirements of the system, and an agile methodology was followed for project management. The result obtained is an efficient and scalable architecture that allows for secure data management and storage and fast and effective processing of user requests.

***Keywords**: Serverless; AWS; GPT, React.*

Índice general

Índice de figuras

1 Introducción

Durante mucho tiempo, a la hora de crear una aplicación web, las arquitecturas que utilizan **servidores físicos** han sido la norma. Esto significa que la aplicación se aloja en un servidor físico, y el servidor es responsable de gestionar todas las peticiones de los usuarios. Este enfoque tiene varias ventajas, como la capacidad de tener más control sobre el entorno de la aplicación, la sencillez de tenerlo todo en un mismo lugar. Sin embargo, también tiene multitud de desventajas, como la necesidad de gestionar el servidor, de mantenerlo actualizado, de pagar por los recursos del servidor sin saber en muchas ocasiones si serán demasiados o escasos, poca capacidad de reaccionar a cambios imprevistos, por nombrar los principales.

En los últimos años, ha surgido un nuevo enfoque para el desarrollo de aplicaciones web: las **arquitecturas sin servidor**. Con las arquitecturas sin servidor, la aplicación no se aloja en un servidor físico. En su lugar, el código de la aplicación se ejecuta en lo que se conoce como la nube, y el proveedor de la nube se encarga de toda la gestión del servidor, las actualizaciones y el escalado. Este enfoque tiene varias ventajas, como la posibilidad de centrarse en el desarrollo de la aplicación sin tener que preocuparse por la infraestructura subyacente, de ampliar o reducir la aplicación automáticamente y la posibilidad de ahorrar dinero en costes de servidor. (Flores, s.f.)

En el proyecto descrito en este documento, se va a utilizar este segundo enfoque para desarrollar una **aplicación web** con la que poder realizar y gestionar recetas de cocina teniendo en cuenta multitud de posibilidades como son las restricciones alimenticias, dificultad de la receta, tiempo, macro nutrientes, etc. Además, el motor encargado de la creación de estas recetas va a ser otra tecnología que se encuentra en alza últimamente, la **inteligencia artificial**, concretamente, el campo de procesamiento de lenguaje natural.

1.1 Motivación

Motivación personal

La motivación para realizar este proyecto ha sido el poder explorar distintos campos de la informática, como son: diferentes **arquitecturas**, usos de la **inteligencia artificial** y **computación en la nube**. Me interesa especialmente la intersección de estos campos y cómo pueden utilizarse para crear aplicaciones web innovadoras y escalables.

En cuanto a la **arquitectura**, me interesa conocer las distintas formas de diseñar y crear aplicaciones web. Quiero entender las ventajas y desventajas de los distintos enfoques y cómo elegir el más adecuado para un proyecto determinado. También me interesa conocer las últimas tendencias en arquitectura web, como los microservicios y la computación sin servidor.

En términos de **inteligencia artificial**, me interesa aprender sobre cómo se puede utilizar la IA para mejorar la experiencia del usuario de las aplicaciones web. Quiero entender cómo utilizar la IA para personalizar el contenido, proporcionar recomendaciones y automatizar tareas. También me interesa conocer los últimos avances en IA, como el aprendizaje profundo y el procesamiento del lenguaje natural.

En cuanto a la **computación en la nube**, me interesa aprender sobre cómo implementar y gestionar aplicaciones web en la nube. Quiero entender los diferentes proveedores de la nube, sus modelos de precios y sus características. También me interesa conocer las últimas tendencias en computación en la nube, como la contenedorización y la computación sin servidor.

Creo que este proyecto me dará la oportunidad de aprender sobre estos diferentes campos y cómo se pueden utilizar para construir aplicaciones web innovadoras y escalables.

Motivación para el lector

Este proyecto también ofrece una motivación importante para el lector interesado en la informática, la inteligencia artificial y la computación en la nube. A través de la exploración de estos campos, el presente trabajo proporciona una visión actualizada de las últimas tendencias y tecnologías utilizadas en el desarrollo de aplicaciones web innovadoras y escalables.

El lector tendrá la oportunidad de adquirir conocimientos en tres áreas fundamentales: arquitectura software, inteligencia artificial y computación en la nube. En cuanto a la arquitectura, se explorarán las opciones disponibles, sus ventajas y desventajas, y las tendencias más recientes como microservicios y computación sin servidor, proporcionando al lector una comprensión sólida en el diseño de arquitecturas web eficientes y flexibles. En el ámbito de la inteligencia artificial, se presentarán formas de implementar los últimos avances para mejorar y dar nuevos servicios en las aplicaciones web. Además, el apartado de computación en la nube ofrecerá al lector una comprensión profunda de la implementación y gestión de aplicaciones web en servicios de proveedores de la nube, incluyendo características clave y tendencias emergentes como la computación sin servidor, prepa-

rando al lector para aprovechar los beneficios de la nube en el desarrollo de aplicaciones web.

En resumen, este proyecto no solo busca satisfacer mi propia motivación en la exploración de estos campos, sino también ofrecer al lector una fuente de inspiración y conocimientos actualizados sobre las últimas tecnologías y tendencias en informática, inteligencia artificial y computación en la nube.

1.2 Objetivos

Diseñar e implementar una arquitectura **serverless** para una aplicación web de creación de recetas basada en Inteligencia Artificial.

- Utilizar una plataforma de computación en la nube para el despliegue de la arquitectura serverless.
- Creación de la aplicación web para la generación de recetas de cocina a partir de las instrucciones del usuario.
- Implementación de un sistema de gestión y almacenamiento de las recetas creadas por los usuarios.
- Integrar una API para generar recetas a partir de las instrucciones del usuario.
- Obtener una arquitectura eficiente y escalable, que permita la gestión y almacenamiento de datos de manera segura y el procesamiento de las solicitudes del usuario de forma rápida y eficaz.

1.3 Estructura

La memoria comienza con una introducción 1 en la que se presenta el proyecto y se establecen los objetivos del mismo. A continuación, se realiza una revisión del estado del arte 2 que incluye una comparativa de los proveedores de servicios en la nube actuales, modelos de lenguaje y una exploración de las principales plataformas desarrollo de aplicaciones web.

En el capítulo 3, se aborda el análisis del problema, donde se presentan los casos de uso y se identifican y analizan las posibles soluciones. Además, se propone una solución específica para resolver el problema planteado.

El diseño de la solución se detalla en el Capítulo 4, donde se describe la arquitectura del sistema y se presenta un diseño detallado. Se especifica también la tecnología utilizada en el desarrollo de la solución.

El desarrollo de la solución propuesta se expone en el Capítulo 5, donde se describe la arquitectura del sistema implementado y se detalla el proceso de desarrollo de la aplicación web.

La implantación del sistema se trata en el Capítulo 6, que incluye los pasos previos necesarios para la puesta en marcha y el despliegue del sistema en producción.

El resultado obtenido se puede ver en Capítulo 7, donde se hace un recorrido por la aplicación desarrollada, explicando las distintas partes de la misma y se ven figuras con el aspecto final.

Finalmente, en el Capítulo 8 se presentan las conclusiones del trabajo, se evalúa si se han alcanzado los objetivos establecidos, los conocimientos adquiridos y la relación del trabajo desarrollado con los estudios cursados. Y en el Capítulo 9 se proponen mejoras que podrían llevarse a cabo en futuras iteraciones del proyecto.

2 Estado del arte

En este capítulo, se abordan tres temas clave en el campo de investigación: las arquitecturas sin servidor, los modelos de lenguaje y el desarrollo de aplicaciones con componentes. Se proporcionará una visión general de cada uno de estos temas, destacando su importancia y presentando ejemplos relevantes. El objetivo es brindar una comprensión clara y práctica de estas áreas de estudio.

2.1 Arquitecturas sin servidor

El concepto de Function-as-a-Service (**FaaS**) o Serverless surge como resultado de la evolución de los modelos de servicio Cloud hacia elementos de computación cada vez más pequeños, donde el proveedor Cloud asume más responsabilidades y el cliente se focaliza en la creación del código para sus aplicaciones.

Antes de la llegada del FaaS, los modelos de computación en la nube más comunes eran **IaaS** (Infrastructure-as-a-Service) y **PaaS** (Platform-as-a-Service). En el modelo IaaS, los proveedores de la nube ofrecían infraestructura virtualizada, como servidores y almacenamiento, permitiendo a los usuarios tener un mayor control y flexibilidad para administrar sus aplicaciones. Por otro lado, en el modelo PaaS, los proveedores de la nube ofrecían una plataforma completa que incluía herramientas y servicios para el desarrollo y despliegue de aplicaciones, ocultando gran parte de la infraestructura subyacente.

Sin embargo, tanto IaaS como PaaS todavía requerían que los desarrolladores gestionaran y escalen sus aplicaciones en función de la carga y los eventos. Fue en este contexto donde surgió el concepto de FaaS o serverless computing. (Deloitte, 2021)

El término "serverless" puede ser un poco engañoso, ya que no significa que no haya servidores involucrados en la ejecución de las aplicaciones. En cambio, se refiere a que los desarrolladores pueden enfocarse exclusivamente en escribir y desplegar funciones individuales, en lugar de preocuparse por la gestión de servidores o infraestructuras subyacentes.

El FaaS permite a los desarrolladores desplegar y ejecutar código en respuesta a eventos específicos, como cambios en los datos, solicitudes HTTP o temporizadores. Los proveedores de la nube se encargan de la infraestructura subyacente, escalando automáticamente los recursos necesarios para ejecutar las funciones de manera eficiente.

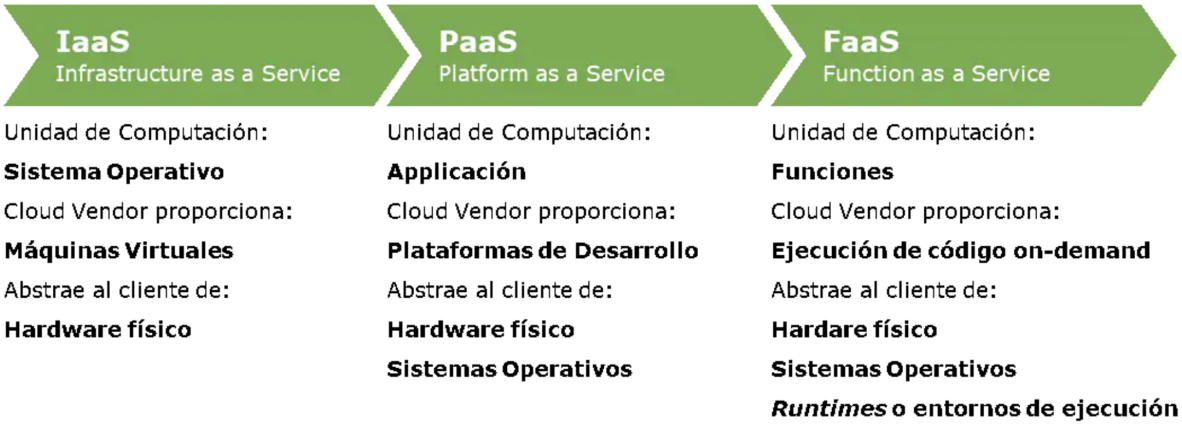


Figura 2.1: Evolución de modelos de computación en la nube

Uno de los primeros servicios comerciales que popularizó el concepto de FaaS fue **AWS Lambda**, lanzado por Amazon Web Services (AWS) en 2014 (Handy, s.f.). AWS Lambda permitía a los desarrolladores ejecutar código en respuesta a eventos sin la necesidad de aprovisionar o gestionar servidores.

Desde entonces, otras plataformas en la nube como Microsoft Azure Functions y Google Cloud Functions han adoptado el enfoque de FaaS, ofreciendo servicios similares a sus clientes.

El FaaS ha ganado popularidad debido a su capacidad para ofrecer una mayor escalabilidad, agilidad y eficiencia en el desarrollo y despliegue de aplicaciones. Los desarrolladores pueden centrarse en escribir código específico de la función, lo que facilita la creación de aplicaciones modulares y altamente distribuidas.

Sin embargo, las arquitecturas sin servidor también presentan algunos desafíos y limitaciones que hay que tener en cuenta.

Beneficios de las arquitecturas sin servidor

Existen varios beneficios al utilizar arquitecturas sin servidor para aplicaciones web. Estos beneficios incluyen:

- **Reducción de los costes de desarrollo y mantenimiento**. El proveedor de la nube se encarga de toda la gestión, las actualizaciones y el escalado del servidor. Esto libera a los desarrolladores para centrarse en el desarrollo de la aplicación, y puede ahorrar dinero en costes de servidor.
- **Escalabilidad automática**. Las arquitecturas sin servidor son automáticamente escalables. Esto significa que la aplicación puede ampliarse o reducirse automáticamente para satisfacer la demanda. Esto puede ahorrar dinero en costes de servidor, al utilizar recursos solo cuando es necesario y mejora la experiencia del usuario garantizando que la aplicación esté siempre disponible.

- **Rendimiento fiable**. El proveedor es el responsable de gestionar la infraestructura subyacente. Y siempre tienen una garantía de rendimiento muy alta gracias a la escala y redundancia, que otorga tranquilidad a los desarrolladores, sabiendo que su aplicación estará disponible cuando los usuarios la necesiten.

- **Rapidez**. Al aprovechar infraestructura ya existente y funcional, se evitan multitud de procesos y esto permite a los desarrolladores iniciar con el desarrollo de aplicaciones desde el primer momento y ser visible mundialmente en poco tiempo.

Desafíos de las arquitecturas sin servidor

Existen algunos retos a la hora de utilizar arquitecturas sin servidor para aplicaciones web. Estos retos incluyen:

- **Coste**. Las arquitecturas sin servidor pueden ser más caras que las arquitecturas tradicionales basadas en servidor. Esto se debe a que el proveedor de la nube cobra por cada petición que se hace a la aplicación. Sin embargo, en muchos casos, el ahorro derivado de la reducción de los costes de desarrollo y mantenimiento puede compensar el aumento del coste de las peticiones.

- **Complejidad**. Las arquitecturas sin servidor pueden ser más complejas que las arquitecturas tradicionales basadas en servidor. Esto se debe a que los desarrolladores tienen que aprender a utilizar las API del proveedor de la nube y a gestionar el estado de la aplicación. Sin embargo, la complejidad de las arquitecturas sin servidor está disminuyendo a medida que la tecnología madura.

- **Dependencia del proveedor**. Cuando se utiliza una arquitectura sin servidor, los desarrolladores suelen estar atrapados en el proveedor de nube que eligen. Esto se debe a que el código de la aplicación se despliega en la infraestructura del proveedor de la nube. Si el desarrollador quiere cambiar de proveedor de nube, tendrá que volver a desplegar el código de la aplicación. Sin embargo, este bloqueo está dejando de ser un problema a medida que más proveedores de nube ofrecen servicios de computación sin servidor.

A medida que el FaaS ha evolucionado, también ha enfrentado desafíos, como la latencia en el inicio en frío de las funciones y la gestión de la comunicación entre funciones. Sin embargo, han surgido servicios para disminuir estos problemas y el FaaS se ha convertido en una parte integral de la arquitectura de muchas aplicaciones modernas y continúa siendo una tendencia en constante desarrollo en la computación en la nube.

2.2 Modelos de lenguaje

Los modelos de lenguaje o procesamiento del lenguaje natural es una rama de la **inteligencia artificial** que se centra en la interacción entre las computadoras y el **lenguaje humano**. El objetivo principal es permitir que las computadoras comprendan, interpreten y generen lenguaje humano de manera efectiva.

Históricamente, el procesamiento del lenguaje natural ha sido un desafío debido a la complejidad y ambigüedad inherentes al lenguaje humano. En las primeras etapas de la investigación, los enfoques se basaban en reglas y patrones predefinidos que los programadores debían crear manualmente. Estos enfoques tenían limitaciones significativas y eran difíciles de escalar debido a la complejidad del lenguaje.

Véase una imagen de la evolución del campo del procesamiento del lenguaje natural en la figura 2.2.

En las últimas décadas, el enfoque ha evolucionado hacia el uso de algoritmos de **aprendizaje automático** y técnicas estadísticas para abordar el procesamiento del lenguaje natural. Uno de los hitos importantes en esta evolución fue el desarrollo de modelos de lenguaje basados en probabilidades, como los modelos de lenguaje basados en N-gramas, que estimaban la probabilidad de ocurrencia de una palabra en función de las palabras anteriores. (Lance, s.f.)

Con el avance del **aprendizaje profundo** y el crecimiento de los conjuntos de datos y la capacidad de cálculo, se han desarrollado modelos más avanzados, conocidos como modelos de lenguaje basados en redes neuronales. Estos modelos utilizan redes neuronales artificiales para aprender patrones y representaciones semánticas del lenguaje. Los modelos más recientes, como **GPT** (Generative Pre-trained Transformer), utilizan arquitecturas de transformers para capturar relaciones a largo plazo en el texto y generar resultados más coherentes y contextuales.

Últimamente estos modelos han avanzado mucho y han demostrado una gran capacidad para generar texto coherente y relevante en una amplia gama de aplicaciones, como asistentes virtuales, traducción automática, generación de contenido y mucho más. Sin embargo, es importante tener en cuenta que estos modelos tienen limitaciones y pueden generar respuestas que no son siempre precisas o adecuadas en todos los contextos. (IBM, s.f.)

2.3 Desarrollo de aplicaciones web

El desarrollo de aplicaciones web ha experimentado una evolución significativa a lo largo de los años.

Las aplicaciones web son programas que se ejecutan en un navegador y permiten a los usuarios acceder a información y servicios a través de Internet. El desarrollo de estas

Figura 2.2: Evolución en el campo de procesamiento del lenguaje natural

aplicaciones ha evolucionado a lo largo del tiempo, pasando por diferentes etapas y tecnologías. (Ríos et al., s.f.)

En los inicios de la web, las aplicaciones se basaban principalmente en **HTML**, **CSS** y **JavaScript**, que son lenguajes que definen la estructura, el estilo y la interacción de las páginas web. Estas aplicaciones eran estáticas, es decir, no podían cambiar su contenido o adaptarse a las preferencias de los usuarios. Además, dependían de la conexión y el rendimiento del servidor para funcionar correctamente.

Para superar estas limitaciones, se empezaron a usar lenguajes de programación del lado del servidor, como **PHP**, **ASP**, **Java** o **Ruby**, que permitían generar páginas web dinámicas, es decir, que podían modificar su contenido según los datos recibidos o enviados por el usuario. Estas aplicaciones se conocen como **web 1.0** o **web tradicional**, y se caracterizan por tener una arquitectura cliente-servidor, donde el cliente (el navegador) solicita una página al servidor y este la envía como respuesta.

Sin embargo, esta arquitectura también presentaba algunos inconvenientes, como la necesidad de recargar la página cada vez que se realizaba una acción, lo que afectaba a la experiencia de usuario y al consumo de recursos. Además, las aplicaciones web 1.0 eran poco interactivas y colaborativas, ya que no permitían la comunicación entre los usuarios o la generación de contenido por parte de estos.

Para mejorar la usabilidad y la funcionalidad de las aplicaciones web, se empezaron a usar tecnologías como **AJAX**, **XML** o **JSON**, que permitían realizar peticiones al servidor sin recargar la página y recibir datos en formatos más ligeros y fáciles de procesar. Estas aplicaciones se conocen como **web 2.0** o **web social**, y se caracterizan por tener una arquitectura cliente-servidor mejorada, donde el cliente (el navegador) puede comunicarse con el servidor de forma asíncrona y recibir solo los datos que necesita para actualizar la página. Además, las aplicaciones web 2.0 son más interactivas y colaborativas, ya que permiten la participación de los usuarios mediante comentarios, valoraciones, etiquetas o redes sociales.

No obstante, esta arquitectura también tiene algunos desafíos, como la complejidad del código, la seguridad de los datos, la compatibilidad entre navegadores o la escalabilidad de las aplicaciones. Para resolver estos problemas, se empezaron a usar herramientas y frameworks que facilitan el desarrollo y el mantenimiento de las aplicaciones web, tanto del

Figura 2.3: Evolución del desarrollo de aplicaciones web

lado del cliente como del servidor. Estas herramientas y frameworks se basan en conceptos como el **MVC** (Modelo-Vista-Controlador), el **REST** (Representational State Transfer) o el **SPA** (Single Page Application).

El MVC es un patrón de diseño que separa las responsabilidades de una aplicación web en tres componentes: el modelo (que gestiona los datos), la vista (que muestra la interfaz de usuario) y el controlador (que coordina las acciones entre el modelo y la vista). El REST es un estilo arquitectónico que define cómo se deben comunicar las aplicaciones web mediante un protocolo estandarizado (HTTP) y un conjunto de operaciones básicas (GET, POST, PUT, DELETE). El SPA es un tipo de aplicación web que carga una sola página en el navegador y actualiza su contenido dinámicamente mediante llamadas al servidor.

Algunos ejemplos de herramientas y frameworks que usan estos conceptos son:

- Del lado del cliente: **Angular**, **React**, **Vue**, **Bootstrap**, etc.
- Del lado del servidor: **Node.js**, **Express**, **Django**, **Laravel**, etc.

Estas herramientas y frameworks se conocen como **web 3.0** o **web semántica**, y se caracterizan por tener una arquitectura cliente-servidor avanzada, donde el cliente (el navegador) tiene más autonomía y responsabilidad en la ejecución de la aplicación, y el servidor se encarga de proveer los datos y los servicios necesarios.

En conclusión, el desarrollo de las aplicaciones web ha experimentado una gran evolución desde sus orígenes hasta la actualidad, pasando por diferentes etapas y tecnologías que han mejorado su diseño, su funcionalidad y su usabilidad. Sin embargo, también se han planteado nuevos retos y oportunidades que requieren de una constante investigación e innovación en este campo.

2.4 Creación de recetas

En la actualidad, la inteligencia artificial (IA) está revolucionando numerosos aspectos de nuestras vidas, y el ámbito de las páginas web no es una excepción. Las aplicaciones de la IA en el desarrollo web han permitido a los usuarios disfrutar de funcionalidades más avanzadas y personalizadas en diversos contextos.

El uso de la inteligencia artificial en el ámbito culinario tiene como objetivo principal satisfacer las necesidades y preferencias individuales de los usuarios, dándoles recetas adaptadas a sus gustos, restricciones dietéticas, alergias alimentarias y otros requisitos específicos. Anteriormente, los usuarios tenían que depender de libros de cocina o buscar recetas en línea que, aunque podían ser útiles, no siempre tenían en cuenta las preferencias y restricciones personales.

Algunos ejemplos de webs que utilizan IA para crear recetas personalizadas son (Geekflare, s.f.):

- **Let's Foodie**: esta web web genera a partir de ciertos ingredientes, explicando paso a paso cómo prepararlas
- **Chefgpt**: también permite generar recetas con lo que se tiene en la nevera, pero además permite seleccionar dificultad, tiempo, etc. Tiene otras funcionalidades más pero son de pago.
- **PlantJammer**: una app vegetariana que usa mapas de sabor y una red neuronal para crear recetas con los ingredientes que se eligen, además de ofrecer consejos y trucos de cocina.

Aunque estas webs son relativamente nuevas y pueden no tener la cantidad de funcionalidades de otro tipo de webs más longevas, demuestran el potencial de la IA para facilitar y mejorar la experiencia culinaria de los usuarios, ofreciendo soluciones creativas, personalizadas y sostenibles. Sin embargo, también plantean algunos desafíos y limitaciones, como por ejemplo:

- La calidad y fiabilidad de las recetas generadas por la IA, que pueden contener errores o inconsistencias.
- La ética y responsabilidad de la IA, que puede influir en las decisiones y hábitos alimenticios de los usuarios.

3 Análisis del problema

Cuando se trata de desarrollar los distintos apartados de un proyecto, existen numerosas soluciones disponibles. En este contexto, resulta fundamental explorar y comprender las diversas alternativas para abordar cada uno de estos aspectos y como queremos resolverlos. En este capítulo, se va a analizar que requisitos funcionales y no funcionales fundamentarán la base del desarrollo de la aplicación.

El **objetivo** principal de este proyecto, por lo tanto, el problema a analizar, es el de crear una aplicación que proporcione a los usuarios una forma sencilla y rápida de **crear recetas** que se ajusten a sus necesidades específicas.

3.1 Casos de uso

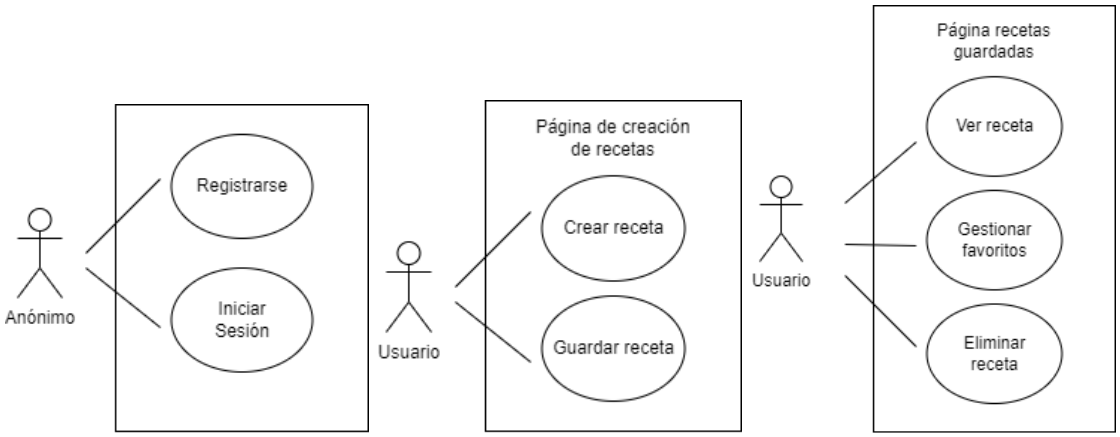
La figura 3.1 muestra los diagramas UML correspondientes a las acciones que se deberían poder realizar en el sistema que da acceso a las funcionalidades relacionadas con las recetas. Este sistema debería permitir distintas funciones en función de los permisos propios del actor, permitiendo el registro y el inicio de sesión a usuarios sin autenticar (anónimos) y la interacción completa con las funcionalidades de crear y gestionar recetas para aquellos autenticados (usuarios).

El diagrama UML muestra las siguientes acciones y relaciones:

- **Registro de usuario**: Permite a un usuario anónimo registrarse en el sistema proporcionando la información requerida, como correo electrónico y contraseña. Esta acción crea una nueva cuenta de usuario y le otorga permisos para acceder a las funcionalidades de crear y gestionar recetas. Para obtener estos permisos el usuario tendrá que verificar el correo electrónico.

- **Inicio de sesión**: Permite a un usuario anónimo iniciar sesión en el sistema utilizando sus credenciales previamente registradas. Al iniciar sesión correctamente, el usuario adquiere los permisos necesarios para acceder a las páginas de creación y gestión de recetas.

- **Crear recetas**: Esta acción está disponible para los usuarios autenticados. Permite a un usuario crear nuevas recetas proporcionando información como: ingredientes,



(a) Usuario anónimo **(b)** Página creación de recetas **(c)** Página recetas guardadas

Figura 3.1: Diagramas de casos de uso

dificultad de preparación, utensilios disponibles, etc. Una vez creada, el usuario podrá guardar la receta en la base de datos, esta estará asociada al usuario que la creó.

- **Ver recetas guardadas**: Esta acción permite a un usuario autenticado acceder a la página que muestra las recetas guardadas. Aquí, el usuario puede ver las recetas que ha creado previamente, así como las recetas que ha marcado como favoritas.

- **Añadir a favoritos**: Dentro de la página de visualización de recetas guardadas, el usuario puede seleccionar una receta y marcarla como favorita. Esta acción permite al usuario guardar las recetas que más le interesen para un fácil acceso posterior.

- **Eliminar recetas**: En la página de visualización de recetas guardadas, el usuario puede eliminar una receta que haya creado previamente. Al seleccionar la opción de eliminación, se elimina la receta correspondiente de la base de datos.

Respecto a la página de creación de recetas, habrá tres posibilidades: despensa, estilo y macros.

- **Despensa**: Esta sección está destinada a listar los ingredientes que tienes disponibles en tu despensa. Puedes ingresar los alimentos que tienes a mano, el tiempo y utensilios disponibles y la página te proporcionará recetas que se puedan preparar con esos ingredientes.

- **Estilo**: En esta sección puedes seleccionar el estilo de cocina que te apetece. Puedes elegir entre diferentes tipos de cocina, como española, china, italiana, mexicana, etc. Si quieres que sea un postre, un desayuno, cena, etc.

- **Macros**: La sección de macros está dirigida a personas que desean tener en cuenta los macronutrientes al planificar sus comidas. Aquí puedes especificar los valores de proteínas, carbohidratos y grasas que deseas incluir en tu receta. La página generará recetas que se ajusten a tus preferencias.

3.2 Requisitos no funcionales

Además de los requisitos funcionales explicados anteriormente, es fundamental considerar una serie de requisitos no funcionales para el desarrollo exitoso de una aplicación, los requisitos más importantes que tendremos en cuenta son:

- **Rendimiento**: La aplicación debe tener un rendimiento rápido y eficiente para garantizar una experiencia fluida del usuario. Esto implica tiempos de carga rápidos, respuestas rápidas a las consultas de los usuarios y capacidad de manejar múltiples solicitudes simultáneamente sin retrasos significativos.

- **Escalabilidad**: La aplicación debe ser capaz de escalar para manejar un crecimiento en la cantidad de usuarios y recetas. A medida que más usuarios utilicen la aplicación, esta debe poder adaptarse y manejar la carga sin degradar el rendimiento.

- **Seguridad**: Dado que la aplicación manejará datos personales de los usuarios, es esencial garantizar la seguridad de la información. Se deben implementar medidas de seguridad adecuadas, como encriptación de datos, autenticación de usuarios y protección contra amenazas como ataques de fuerza bruta o acceso no autorizado.

- **Disponibilidad**: La aplicación debe estar disponible de manera confiable y tener una alta disponibilidad para los usuarios. Esto implica minimizar los tiempos de inactividad planificados y no planificados, y tener un sistema de copias de seguridad y recuperación de datos para mitigar cualquier pérdida o interrupción en el servicio.

- **Usabilidad**: La aplicación debe ser intuitiva y fácil de usar, incluso para usuarios sin experiencia técnica. La interfaz de usuario debe ser clara, bien organizada y permitir a los usuarios navegar sobre los diferentes apartados de manera sencilla. Además, se debe considerar el diseño responsive para adaptarse a diferentes dispositivos y tamaños de pantalla.

- **Adaptabilidad**: La aplicación debe ser flexible y adaptable a diferentes preferencias y restricciones alimentarias. Esto implica la capacidad de personalizar las recomendaciones y sugerencias según las preferencias individuales de cada usuario, teniendo en cuenta alergias, restricciones dietéticas y gustos personales.

- **Mantenibilidad**: La aplicación debe ser fácil de mantener y actualizar. Se deben seguir buenas prácticas de desarrollo de software y utilizar una arquitectura modular que facilite la incorporación de nuevas características, la corrección de errores y la mejora continua de la aplicación.

4 Diseño de la solución

Una vez analizado los requisitos de la aplicación a desarrollar, el siguiente paso consiste en decidir qué tecnologías y herramientas se utilizarán en el desarrollo de la aplicación y en diseñar la arquitectura y la interfaz de la aplicación. En este capítulo se definirán los componentes principales del sistema, cómo se comunicarán entre sí y cómo se integrarán las funcionalidades. Además, se establecerá el flujo de la interfaz de usuario, asegurando una experiencia intuitiva y amigable para los usuarios. También se abordarán aspectos como la seguridad, el rendimiento y la escalabilidad del sistema, garantizando que la aplicación cumpla con los requisitos y expectativas establecidos.

4.1 Identificación y análisis de soluciones posibles

La aplicación contará con un **frontend** intuitivo y fácil de usar, donde los usuarios podrán ingresar sus preferencias de ingredientes, restricciones dietéticas, tiempo de preparación y otros parámetros relevantes. Estos datos serán enviados al backend de la aplicación para su procesamiento.

El **backend** será responsable de manejar la lógica de negocio y la interacción con los servicios externos. Comprobar si el usuario está autenticado, almacenar recetas en la base de datos, devolverlas, entre otras funciones.

Una vez que los inputs del usuario se hayan recopilado, el backend enviará las consultas a una **API** de procesamiento de lenguaje natural. Esta API será responsable de analizar y comprender las preferencias del usuario, así como de generar recomendaciones de recetas basadas en esa información.

Una vez que las recomendaciones se generen, se mostrarán al usuario a través del frontend de la aplicación. El usuario podrá ver los detalles de las recetas sugeridas, como los ingredientes, las instrucciones de preparación y la información nutricional. Además, se le proporcionará la opción de guardar las recetas que le resulten interesantes para acceder a ellas más adelante.

En resumen, el objetivo es diseñar e implementar una arquitectura serverless para una aplicación de creación de recetas personalizadas. La arquitectura permitirá recopilar los

inputs del usuario a través de un frontend, enviarlos al backend para su procesamiento, utilizar una API de procesamiento de lenguaje natural para generar recomendaciones de recetas y mostrarlas al usuario a través del frontend. La aplicación también brindará la opción de guardar las recetas para un acceso posterior. Esta solución serverless ofrecerá escalabilidad, eficiencia y facilidad de gestión, lo que proporcionará una experiencia de usuario óptima en la creación de recetas personalizadas.

4.1.1 Front-end

HTML, CSS y JavaScript puros: Esta es la forma más básica de crear un frontend. Utilizando HTML para estructurar el contenido y CSS para estilizarlo y JavaScript para añadir funcionalidades, permite tener un control total sobre el código, pero también requiere más tiempo y esfuerzo para crear y mantener el frontend.

Frameworks y bibliotecas de JavaScript: Existen varios proyectos que facilitan la construcción de frontends más complejos y sofisticados. Algunos ejemplos populares son React, Angular y Vue.js. Estas herramientas proporcionan una estructura y una sintaxis más organizada para desarrollar aplicaciones web interactivas. Además, suelen ofrecer funcionalidades adicionales, como enrutamiento, gestión de estado y manipulación eficiente del DOM.

Generadores de sitios estáticos: Los generadores de sitios estáticos, como Jekyll, Hugo o Gatsby, permiten construir frontends utilizando plantillas y contenido preprocesado. Estas herramientas generan una versión estática del sitio web que se puede implementar fácilmente en un servidor. Son ideales para sitios web con contenido principalmente estático y que no requieren mucha interactividad. (Cloudflare, s.f.)

Sistemas de Gestión de Contenidos (CMS): Los CMS, como WordPress o Drupal, ofrecen una interfaz visual para crear y administrar contenido no solo frontend, si no también backend. Estas plataformas permiten personalizar la apariencia del sitio web mediante temas y complementos, y suelen tener funcionalidades integradas, como gestión de usuarios, SEO y comercio electrónico. Como contra, estas plataformas también pueden presentar algunas limitaciones en cuanto a flexibilidad y rendimiento en comparación con el desarrollo personalizado de aplicaciones web. Además, a medida que las necesidades y requisitos del proyecto se vuelven más complejos, los CMS pueden requerir un mayor esfuerzo de personalización y adaptación para satisfacer esas demandas específicas.

4.1.2 Back-end

Monolito: En esta arquitectura, todo el backend se desarrolla como un solo componente o aplicación. Todas las funciones y características están integradas en un único código base. Es fácil de desarrollar y probar, pero puede volverse complicado y difícil de escalar a medida que el proyecto crece.

Arquitectura orientada a servicios (SOA): En esta arquitectura, el backend se divide en servicios independientes que se comunican entre sí a través de interfaces bien definidas.

Cada servicio se centra en una función específica y puede ser desarrollado, probado y escalado de forma independiente. Es más modular y escalable que el enfoque monolítico.

Microservicios: Es una evolución de la arquitectura SOA. En lugar de servicios grandes, se construyen servicios más pequeños y autónomos, cada uno con su propia lógica de negocio y base de datos. Los microservicios se comunican entre sí a través de API y pueden ser desarrollados, probados y desplegados de forma independiente. Esta arquitectura permite una mayor escalabilidad y flexibilidad, pero también agrega complejidad adicional en cuanto a la gestión de la comunicación entre los servicios.

Contenedores y orquestadores: Con esta opción, el backend se divide en pequeños contenedores, que encapsulan el código y las dependencias necesarias para su ejecución. Luego, se utiliza un orquestador de contenedores, como Kubernetes, para gestionar, escalar y distribuir estos contenedores en un clúster de servidores. Permite una mayor flexibilidad y escalabilidad, pero también agrega complejidad adicional en cuanto a la configuración y administración de los contenedores y el orquestador.

Serverless: Esta arquitectura elimina la necesidad de administrar servidores. El código del backend se ejecuta en funciones individuales (llamadas funciones sin servidor o "serverless") que se ejecutan en la nube y se escalan automáticamente según la demanda. Los proveedores de servicios en la nube, como AWS Lambda, Azure Functions o Google Cloud Functions ofrecen entornos para desarrollar y desplegar estas funciones. Es altamente escalable, pero puede tener limitaciones en cuanto a la duración de las funciones y el tiempo de ejecución.

4.1.3 Generación de recetas

Este es el apartado más reciente, y aunque están surgiendo constantemente nuevas opciones y modelos de Procesamiento del Lenguaje Natural, actualmente la oferta de APIs es limitada.

Una de las opciones más destacadas en la generación de texto con IA es el modelo de lenguaje **GPT** desarrollado por **OpenAI**.

Además de GPT, existen otros modelos de lenguaje pre-entrenados que también pueden generar texto de manera efectiva. Por ejemplo, BERT (Bidirectional Encoder Representations from Transformers) es otro modelo destacado en el campo del NLP. Aunque su enfoque principal es el de comprensión del lenguaje, también se puede utilizar para generar texto (Martínez, s.f.) (Devlin & Chang, s.f.).

En cuanto a las APIs y plataformas disponibles para generar texto con IA, algunas opciones populares son:

OpenAI API: OpenAI ofrece una API que permite a los desarrolladores acceder a modelos de lenguaje de vanguardia, como GPT-3. A través de la API, se puede enviar un texto de entrada y obtener una respuesta generada por el modelo.

Hugging Face Transformers: Hugging Face es una plataforma que proporciona acceso a una amplia gama de modelos de lenguaje pre-entrenados, incluidos GPT-2, GPT-3, BERT y muchos más. Ofrece una API fácil de usar que permite generar texto utilizando estos modelos.

Google Cloud Natural Language API: Google Cloud ofrece una API de lenguaje natural que brinda capacidades de generación de texto junto con otras funcionalidades de procesamiento de lenguaje. Esta API permite generar texto a partir de una serie de comandos y parámetros.

Microsoft Azure Text Analytics: Microsoft Azure ofrece servicios de análisis de texto que incluyen generación de texto. La API de Text Analytics permite generar resúmenes, respuestas a preguntas y texto personalizado utilizando modelos pre-entrenados.

Es importante tener en cuenta que estas opciones están en constante evolución, y seguramente nuevas APIs y plataformas se hayan lanzado desde la última actualización de esta información. Sin embargo, las mencionadas anteriormente son algunas de las opciones más populares y probablemente se actualizarán a los cambios.

4.2 Solución propuesta

Teniendo ya conocimiento de que tecnologías existen, su evolución y diferentes plataformas, se va explicar cuales se van a usar en este proyecto y porque.

En este trabajo, se utilizará **AWS** (Amazon Web Services) para montar la arquitectura serverless. AWS es reconocida como una de las plataformas de computación en la nube más populares y ampliamente utilizadas en la industria. La elección de AWS se debe a su extensa documentación, infraestructura robusta y una amplia gama de servicios que cubren todas las necesidades del proyecto.

Además, se utilizará la **API** de **ChatGPT**, una de las opciones más destacadas y que mejores resultados está aportando en la generación de respuestas de lenguaje natural en la actualidad. La API de ChatGPT ofrece una interfaz sencilla y completa para interactuar con el modelo de lenguaje IA de ChatGPT, evitando la necesidad de desarrollar y entrenar un modelo propio desde cero. Esto permite integrar capacidades de procesamiento de lenguaje natural avanzadas en la aplicación de forma eficiente.

Por otro lado, para la parte web de la aplicación, se empleará **React**, un framework de JavaScript ampliamente utilizado para desarrollar interfaces de usuario interactivas. React ofrece una forma estructurada y eficiente de crear componentes reutilizables y mantener un flujo de datos consistente. Además, cuenta con una comunidad de desarrolladores activa y una amplia selección de bibliotecas y complementos que facilitan el desarrollo web.

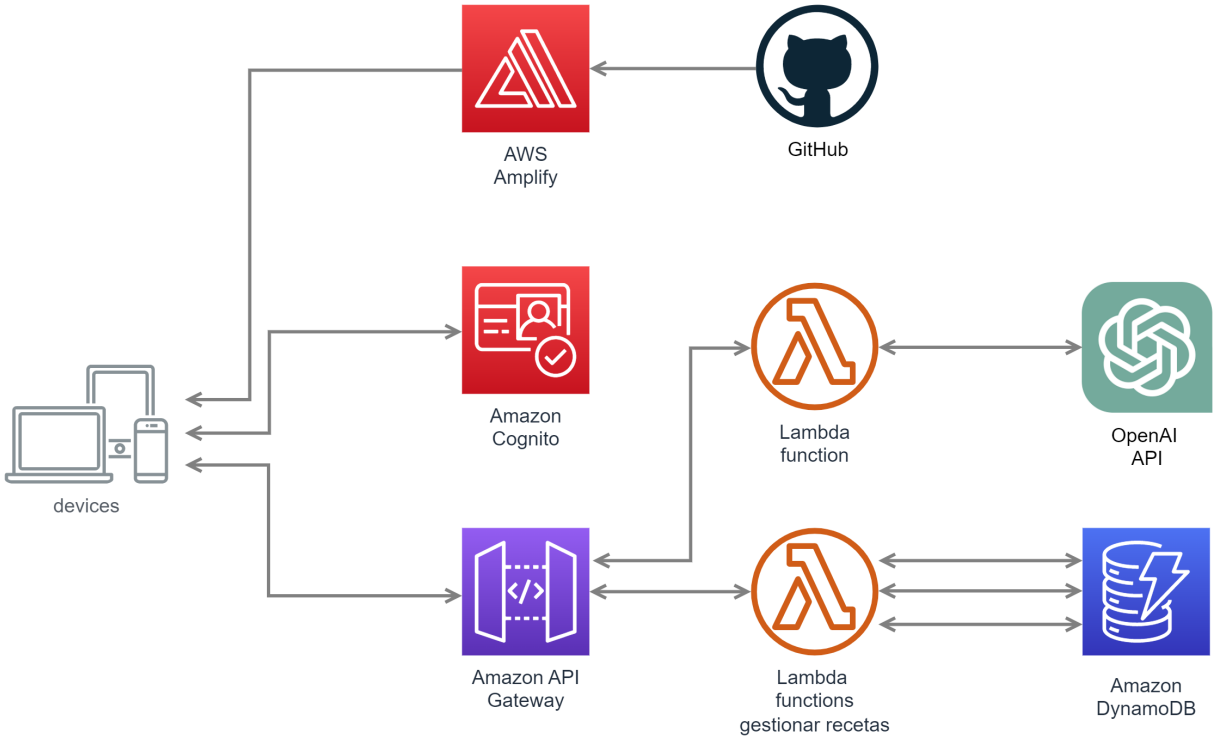


Figura 4.1: Esquema general de la arquitectura del sistema

4.3 Arquitectura propuesta

El esquema general de la arquitectura se puede observar en la 4.1. Una explicación rápida de que hace cada componente es:

- **GitHub** será el servicio encargado de almacenar el código fuente y control de versiones de la aplicación.
- **OpenAI API** el servicio encargado de generar las recetas personalizadas se llamará a través de su API.
- **AWS Amplify** será el servicio encargado de hostear la aplicación web y aplicar el despliegue e integración continuos para mantener la página siempre operativa.
- **Amazon Cognito** se encargará de llevar toda la parte de autenticación y registro de usuarios.
- **Amazon API Gateway** será la puerta de entrada a las peticiones del usuario al backend.
- **AWS Lambda** serán las funciones encargadas de llevar las labores de backend, crear recetas, gestionarlas, etc.
- **Amazon DynamoDB** será la base de datos encargada de almacenar las recetas y usuarios.

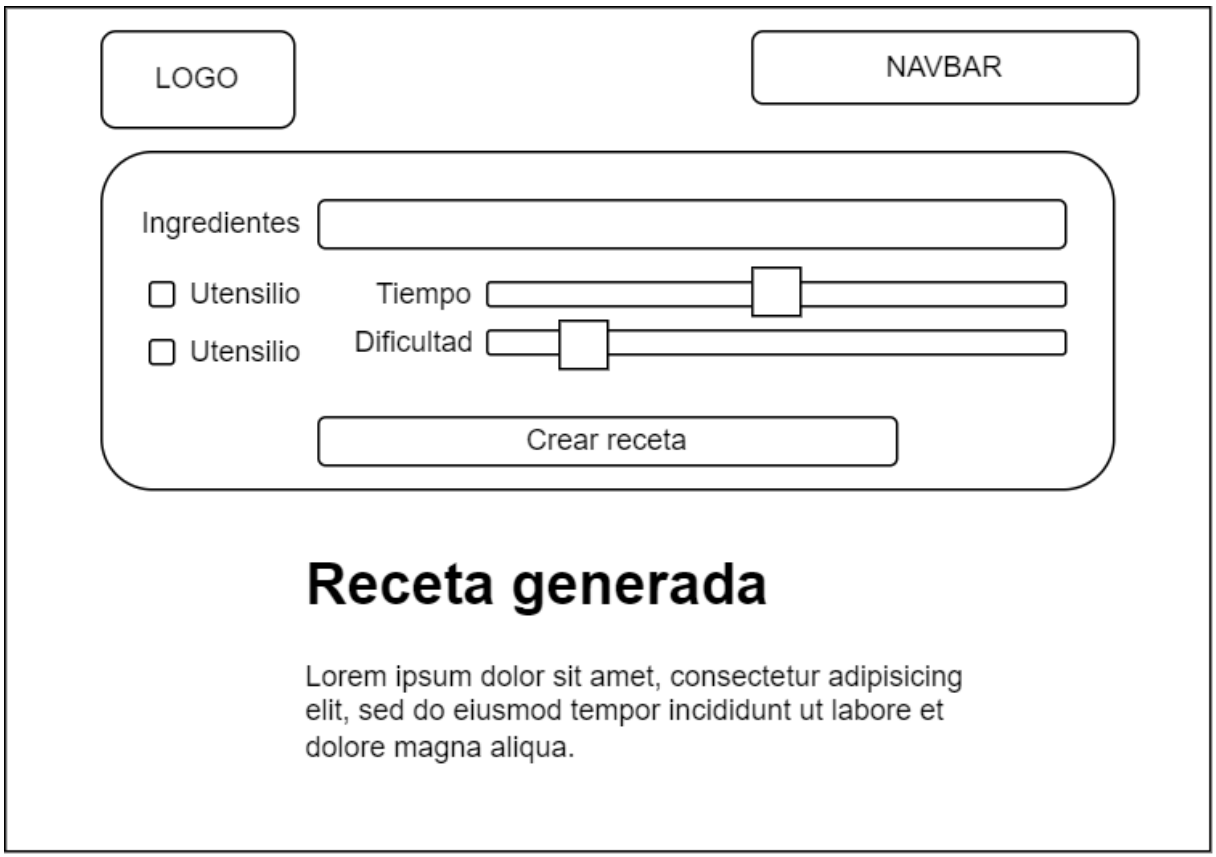


Figura 4.2: Mockup del diseño web

4.4 Diseño web

Se abordará la creación de una interfaz atractiva y funcional para la aplicación. Se prestará especial atención al diseño de la interfaz de usuario (UI) y la experiencia del usuario (UX). Se considerarán aspectos como la disposición de elementos, la elección de colores, la legibilidad del texto y la navegación intuitiva. Se tiene que poder navegar a la sección que quieras fácilmente y poder crear tu receta personalizada con pocos clicks y sin cambiar de página, un mockup de esto se puede ver en la figura 4.2. El objetivo es crear una interfaz que sea agradable a la vista, fácil de usar y proporcione una experiencia positiva para los usuarios.

4.5 Tecnología Utilizada

En resumen, para el desarrollo de este proyecto se han utilizado diversas tecnologías y herramientas para implementar de manera efectiva cada componente del sistema. En el capitulo 5 se explicará mas en profundidad cada una de estas pero a continuación, se resumen las principales tecnologías empleadas:

- **Control de Versiones**: Se utilizarán Git y GitHub para el control de versiones del código.

- **React**: Un framework de JavaScript, para crear una interfaz de usuario interactiva y dinámica.
- **Bootstrap**:Un framework de CSS, para agilizar el diseño y los estilos de la interfaz.
- **Node.js**: Entorno de ejecución JavaScript para el código backend de las Lambdas.
- **OpenAI API**: API de OpenAI, en particular el modelo GPT-3.5 Turbo, para generar texto con inteligencia artificial.
- **Amazon Cognito**: Para gestionar la autenticación y autorización de los usuarios de manera segura y escalable.
- **API Gateway**: Como punto de entrada de las solicitudes del cliente, facilitando la gestión y el enrutamiento adecuado de las peticiones a los servicios correspondientes.
- **AWS Amplify**: Despliegue de la aplicación web y facilitar la integración con otros servicios.
- **AWS Lambda**: Para implementar funciones sin necesidad de gestionar la infraestructura subyacente.
- **Amazon DynamoDB**: Base de datos no relacional para almacenar las recetas y otra información relevante del sistema.
- **AWS SAM (Serverless Application Model)**: Para definir y desplegar la infraestructura como código, transformando las definiciones en plantillas de AWS CloudFormation.
- **AWS CloudWatch**: Recopilar y visualizar los registros, las métricas y los datos de evento en tiempo real para observar logs y mejorar el mantenimiento de la aplicación.

5 Desarrollo de la solución propuesta

En este capítulo, se describe el proceso de desarrollo de la solución propuesta a partir del diseño previo. Se detalla cómo se llevará a cabo el desarrollo de la arquitectura y las aplicaciones, y se explican las tecnologías y metodologías utilizadas. El objetivo es proporcionar una visión clara y práctica de la implementación de la solución propuesta.

5.1 Arquitectura del Sistema

Lo primero que vamos a tener que decidir ahora que tenemos clara que arquitectura queremos montar, se puede ver en la figura 4.1, es como construirla.

En Amazon Web Services (AWS), existen varias formas de montar una arquitectura, las opciones más comunes son:

AWS Console: Usar la consola web de AWS es una de las formas más comunes y accesibles para montar una arquitectura en AWS. La consola web proporciona una **interfaz gráfica** intuitiva que te permite administrar y configurar tus recursos de AWS sin necesidad de escribir código o utilizar herramientas adicionales. Es lo primero que se recomienda para empezar a probar los diferentes servicios y manejarlos una vez ya están creados. Pero no es una buena forma de montar una arquitectura ya que todo lo que se haga con esta opción es manual y pueden desencadenarse errores.

AWS Command Line Interface (CLI): AWS CLI es una interfaz de **línea de comandos** que te permite interactuar con los servicios de AWS desde la línea de comandos. Puedes utilizar comandos predefinidos para crear y configurar recursos de AWS, así como automatizar tareas en tu infraestructura. Es una opción muy flexible y poderosa para aquellos que prefieren trabajar desde la línea de comandos.

AWS SDK para diferentes **lenguajes de programación**: AWS proporciona SDKs para varios lenguajes de programación populares, como Python, Java, .NET, Ruby, entre otros. Estos SDKs te permiten interactuar con los servicios de AWS directamente desde tu aplicación. Puedes utilizar el SDK correspondiente a tu lenguaje de programación preferido para crear, configurar y administrar recursos de AWS en tu aplicación.

AWS CloudFormation: CloudFormation es un servicio de AWS que te permite crear y administrar recursos de AWS utilizando plantillas declarativas. Con CloudFormation, puedes describir tu **infraestructura como código** en un archivo de **plantilla** y luego implementar y actualizar esa infraestructura de manera consistente y repetible. Es una forma eficiente de automatizar la creación y administración de recursos de AWS.

5.1.1 AWS SAM

Teniendo en cuenta diversas alternativas disponibles en AWS para la construcción de una arquitectura, a continuación presento de manera más detallada la opción que he elegido utilizar y como es el flujo de trabajo.

AWS SAM (**Serverless Application Model**) es un marco de desarrollo de aplicaciones sin servidor que simplifica la creación, implementación y administración de aplicaciones sin servidor en Amazon Web Services (AWS). Proporciona una **sintaxis simplificada** y una serie de **herramientas** para **definir** y **desplegar** aplicaciones sin servidor utilizando servicios de AWS, como AWS Lambda, API Gateway y DynamoDB, entre otros.

La principal idea detrás de AWS SAM es permitir a los desarrolladores describir su infraestructura y lógica de aplicación de manera declarativa utilizando un archivo de plantilla **YAML** o JSON. Esto permite una gestión simplificada de la infraestructura y reduce la complejidad del despliegue y la configuración de las aplicaciones sin servidor.

AWS SAM se basa en CloudFormation, el servicio de AWS para la orquestación y gestión de recursos de infraestructura. Utiliza la misma sintaxis de plantillas de CloudFormation, pero se centra específicamente en la creación de aplicaciones sin servidor.

El flujo de trabajo típico con AWS SAM implica los siguientes pasos:

- **Definición de plantilla**: Se crea un archivo de plantilla YAML o JSON que describe la infraestructura necesaria y la lógica de la aplicación.
- **Desarrollo de la función**: Se desarrolla el código de la función de AWS Lambda utilizando el lenguaje de programación compatible, como Node.js, Python o Java.
- **Prueba local**: Se prueba la función de AWS Lambda localmente utilizando el SAM CLI (Command Line Interface) para garantizar que funciona correctamente.
- **Implementación**: Se utiliza el SAM CLI para empaquetar la aplicación y los recursos en un archivo ZIP y se implementa en AWS utilizando CloudFormation.

AWS SAM además proporciona comandos y herramientas adicionales a través del SAM CLI para facilitar la creación y gestión de aplicaciones sin servidor.

Algunos de los comandos más utilizados son:

- **sam init**: Crea una nueva aplicación sin servidor basada en plantillas predefinidas. Puedes seleccionar un proyecto de ejemplo o una plantilla vacía y especificar el lenguaje de programación que deseas utilizar.

- **sam build**: Construye la aplicación sin servidor y sus dependencias. Este comando resuelve las dependencias y empaqueta el código fuente en un archivo ZIP que se utilizará para la implementación.

- **sam local invoke**: Invoca la función de AWS Lambda localmente utilizando datos de prueba. Este comando te permite probar y depurar la función sin necesidad de implementarla en AWS.

- **sam local start-api**: Inicia una API Gateway localmente para probar y depurar tus funciones de AWS Lambda que se integran con ella. Puedes enviar solicitudes HTTP a la API local y obtener respuestas simuladas de tus funciones.

- **sam deploy**: Implementa la aplicación en AWS utilizando CloudFormation. Este comando crea o actualiza los recursos necesarios y despliega la aplicación sin servidor.

5.1.2 Amazon Cognito

La autenticación es el proceso de verificación de la identidad de un usuario. Es importante implementar la autenticación para proteger los datos de los usuarios y evitar accesos no autorizados, además de que usaremos esta identidad para poder tener un inventario de recetas guardadas y favoritas.

Existen diferentes formas de implementar la autenticación en las aplicaciones web. Un método común es utilizar un nombre de usuario y una contraseña. Otro método es utilizar un servicio de autenticación de terceros, como Google o Facebook.

En este proyecto, utilizaremos **Amazon Cognito** para gestionar la autenticación de los usuarios. Esta herramienta proporciona un sistema de autenticación estándar, usuario y contraseña, donde se pueden modificar ciertos parámetros para adaptarlos a las necesidades específicas del proyecto, como longitud mínima de contraseñas, etc.

Utilizaremos un sistema de autenticación basado en **email y contraseña**, el cual además requerirá la **verificación** del correo electrónico.

Para ello se va a explicar como funciona este servicio y como lo implementaremos con AWS SAM.

Amazon Cognito funciona con **grupos de usuarios**. Los grupos de usuarios son una funcionalidad que te permite organizar a los usuarios en conjuntos lógicos dentro de tu aplicación o servicio web. Los grupos se utilizan para asignar permisos y acceder a diferentes recursos según las necesidades de tu aplicación. Estos grupos pueden ser administrados y controlados de forma centralizada a través de Amazon Cognito.

En Amazon Cognito, las agrupaciones de usuarios se organizan en conjuntos llamados "**pools**" o piscinas. Una "pool de usuarios" es un contenedor para los usuarios registrados en tu aplicación. Cada grupo de usuarios pertenece a una piscina específica. Puedes crear varias piscinas de usuarios en una cuenta de Amazon Cognito y cada piscina puede tener diferentes configuraciones y propiedades.

Cuando creas una piscina de usuarios, puedes definir atributos personalizados para los usuarios, como su nombre, dirección de correo electrónico, número de teléfono, etc. Estos atributos pueden ser utilizados para segmentar y administrar los grupos de usuarios de manera más efectiva.

Una vez que tienes un pool de usuarios creado y configurado, puedes empezar a agregar grupos de usuarios a esa piscina. Por ejemplo, puedes tener un grupo de usuarios llamado "Administradores" que tenga permisos de administración completos, y otro grupo llamado "Usuarios" que tenga permisos limitados. Puedes asignar diferentes permisos y políticas de acceso a cada grupo, lo que permite un control más granular sobre qué recursos y funcionalidades pueden utilizar los usuarios en cada grupo.

Cuando un usuario se registra en tu aplicación y se autentica a través de Amazon Cognito, se le asigna automáticamente al grupo predeterminado de la piscina de usuarios. Sin embargo, puedes modificar la pertenencia del usuario a diferentes grupos según tus necesidades. Esto te permite otorgar y revocar privilegios fácilmente, así como personalizar la experiencia y los permisos para diferentes conjuntos de usuarios.

En resumen, los grupos de usuarios en Amazon Cognito te permiten organizar y administrar a tus usuarios en conjuntos lógicos dentro de tu aplicación. Puedes asignar permisos y políticas de acceso diferentes a cada grupo, lo que te da un mayor control sobre los recursos y funcionalidades a los que pueden acceder los usuarios. Las "pools" son contenedores que albergan los grupos de usuarios y permiten gestionar y configurar propiedades específicas para cada conjunto de usuarios dentro de tu aplicación.

En nuestro caso, crearemos un único "pool de usuarios" llamado "**RecipeUserPool**". Esto se puede crear en la plantilla YAML de SAM de la siguiente forma:

```
RecipeUserPool:
    Type: AWS::Cognito::UserPool
    Properties:
      AdminCreateUserConfig:
        AllowAdminCreateUserOnly: false
      UserPoolName: RecipeUsers
      UsernameAttributes:
      - email
      AutoVerifiedAttributes:
      - email
      Policies:
        PasswordPolicy:
          MinimumLength: 6
          RequireLowercase: true
          RequireNumbers: false
          RequireSymbols: false
          RequireUppercase: true
```

Además del "pool de usuarios", hay que crear un "**UserPoolClient**", que será el encargado de actuar como un intermediario entre la aplicación cliente y el User Pool.

```
RecipeUserPoolTokenClient:
```

```
    Type: AWS::Cognito::UserPoolClient
    Properties:
      UserPoolId: !Ref RecipeUserPool
      GenerateSecret: false
      AllowedOAuthFlowsUserPoolClient: true
      AllowedOAuthFlows: ['code', 'implicit']
      CallbackURLs: ['http://localhost:3000', ...]
      SupportedIdentityProviders: ['COGNITO']
      AllowedOAuthScopes: ['phone', 'email', 'openid']
      ExplicitAuthFlows:
      - USER_PASSWORD_AUTH
```

Al crear un User Pool Client, se obtiene un conjunto de credenciales (ID de cliente y secreto de cliente) que se usarán en la aplicación para realizar operaciones de autenticación y autorización, registro, etc.

5.1.3 API Gateway

Teniendo el "pool" de usuarios, procederemos a crear la API Gateway, la cual actuará como **intermediario** para conectar con las funciones lambda. Nuestro objetivo es restringir el acceso a esta API solo a usuarios autenticados mediante el grupo de usuarios mencionado anteriormente.

Para ello los pasos principales para desarrollar esta API Gateway son:

- **Definición de API**: Primero, se define la API mediante la configuración de endpoints, métodos HTTP permitidos y los recursos disponibles.
- **Integración de servicios**: Luego, se integran los servicios subyacentes, como AWS Lambda, Amazon DynamoDB u otros servicios web externos, en la API Gateway. Esto permite que la API Gateway enrute las solicitudes de los clientes a los servicios correspondientes.
- **Configuración de políticas de seguridad**: Se establecen políticas de seguridad para proteger la API y controlar el acceso. Esto puede incluir la autenticación de usuarios, autorización basada en roles, la generación de tokens de acceso, el uso de certificados SSL/TLS, entre otros mecanismos de seguridad.

Además la API Gateway también se encarga de: transformar los datos si es necesario, entrega de respuestas, monitoreo y análisis entre otras funciones que se pueden ajustar dependiendo de las necesidades.

Para manejar las operaciones relacionadas con las recetas, se establecerá un endpoint utilizando el path "/**item**". A través de este endpoint, se podrán realizar diferentes acciones utilizando los métodos **HTTP** estándar.

- Para añadir una nueva receta, se utilizará el método **POST**. Los clientes podrán enviar los datos de la receta a través de una solicitud POST al endpoint "/item".

- Para obtener una lista de recetas existentes, se utilizará el método GET. Al realizar una solicitud **GET** al endpoint "/item", el API Gateway redirigirá la petición a la Lambda correspondiente, que devolverá la lista de recetas.

- Para eliminar una receta específica, se utilizará el método **DELETE**. Los clientes podrán enviar una solicitud DELETE al endpoint "/item/id", donde "id" representa el identificador único de la receta que se desea eliminar. El API Gateway redirigirá la petición a la Lambda correspondiente encargada de eliminar la receta.

Además del path mencionado anteriormente, existirán otros path que brindarán funcionalidades adicionales:

- Para añadir una receta a la lista de favoritos, se utilizará el endpoint "**/item/addFav**". Mediante una solicitud POST a este endpoint, los clientes podrán enviar los datos de la receta que deseen marcar como favorita.

- Para solicitar la generación de recetas personalizadas, se utilizará el endpoint "**/askChat**". Mediante una solicitud POST a este endpoint, los clientes podrán enviar los criterios o preferencias para la generación de recetas.

Para definir la API en el formato YAML de SAM se haría de la siguiente forma:

```
RecipeApi:
    Type: AWS::Serverless::Api
    MethodSettings:
        DataTraceEnabled: true
        MetricsEnabled: true
        HttpMethod: '*'
        ResourcePath: !Sub '${VersionParam}/*'
        LoggingLevel: INFO
    Properties:
      Name: RecipeApi
      StageName: !Ref StageNameParam
      TracingEnabled: true
      Cors:
        AllowOrigin: "'*'"
        AllowMethods: "'OPTIONS,HEAD,GET,PUT,POST,DELETE'"
        AllowHeaders: "'Content-Type,X-Amz-Date,Authorization
           ...'"
      Auth:
        Authorizers:
          CognitoAuthorizer:
            UserPoolArn: !GetAtt "RecipeUserPool.Arn"
```

Se puede observar como se hace uso del UserPool creado anteriormente para de esta forma decir que la API sólo pueda ser usada por los usuarios autentificados.

5.1.4 Lambdas

Hemos creado la API Gateway, esta se encargará de hacer de intermediaria con las funciones Lambdas, pero para ello hay que ver como funciona AWS Lambda y como hacer que la API accione estas funciones.

Para crear una función Lambda hay que seguir los siguientes pasos:

Carga y configuración: El desarrollador carga su código en AWS Lambda y especifica la configuración necesaria, como el lenguaje de programación, la memoria asignada, el tiempo de espera, las variables de entorno, entre otros parámetros.

Evento de activación: AWS Lambda se configura para responder a eventos específicos, como una solicitud HTTP, una inserción en una cola de mensajes o una modificación en un bucket de Amazon S3. Cuando ocurre un evento, se activa la ejecución de la función Lambda asociada.

Asignación de recursos: AWS Lambda asigna automáticamente recursos de cómputo, como CPU y memoria, para ejecutar la función Lambda. Estos recursos se escalan automáticamente según la demanda, pero se puede configurar que empiece con unos determinados recursos, poner límites, etc.

Configuración de permisos y acceso: Antes de que la función Lambda pueda acceder a los recursos de AWS, es necesario configurar las políticas de permisos adecuadas. Esto se logra mediante la asignación de roles de IAM (Identity and Access Management) a la función Lambda. Estos roles de IAM especifican qué servicios y recursos puede acceder la función Lambda y qué acciones puede realizar.

Para crearla, la estructura en YAML siguiendo las instrucciones de AWS SAM es:

```
GetRecipeFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: recipe-src/getRecipe
      Handler: app.getRecipeItem
      Tracing: Active
      Policies:
        - DynamoDBReadPolicy:
            TableName: !Ref RecipeTable
        - CloudWatchPutMetricPolicy: {}
      Environment:
        Variables:
          TABLE_NAME: !Ref RecipeTable
          AWS_NODEJS_CONNECTION_REUSE_ENABLED: "1"
          USE_DYNAMODB_LOCAL: "0"
          DYNAMODB_LOCAL_URI: ""
      Events:
        GetItem:
          Type: Api
          Properties:
            Path: /item/{id}
```

```
            Method: get
            RestApiId: !Ref RecipeApi
            Auth:
              Authorizer: CognitoAuthorizer
```

Podemos ver como esta funcion Lambda:

- Ejecutara el código que se encuentra en "recipe-src/getRecipe".
- Tendrá permisos para leer en la base de datos.
- Estará reaccionando a un evento de la API Gateway, en concreto, una llamada HTTP GET al endpoint "/item/id", que tendrá que estar autorizado con Cognito en el pool creado anteriormente.

El código que se encontrará en el directorio "recipe-src/getRecipe" consistirá en un archivo "app.js" de Node.js, el cual permitirá desarrollar la lógica que deseemos. Sin embargo, es importante tener en cuenta que el punto de entrada de ejecución será el dado por el "Handler" definido como:

```
exports.getRecipeItem = async function (event, context) {
  //Punto de entrada
};
```

Es importante saber que el ciclo de vida de una función lambda se compone de tres fases distintas: "**init**", "**invoke**" y "**shut down**".

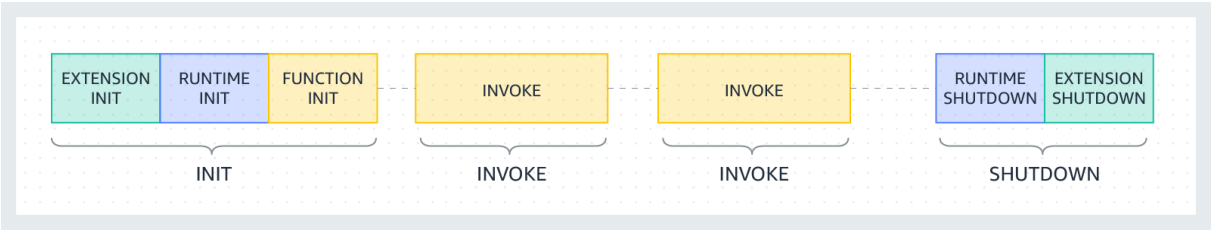


Figura 5.1: Ciclo de vida de una función Lambda

En la fase "**init**", Lambda comienza todas las extensiones, inicia el tiempo de ejecución y ejecuta el código estático de la función. Esta fase finaliza cuando el tiempo de ejecución y todas las extensiones señalan que están listas mediante el envío de una solicitud Next a la API.

En la fase "**invoke**", Lambda invoca la función y el código de la extensión en respuesta a los eventos activadores. Esta fase finaliza cuando el tiempo de ejecución y todas las extensiones se han completado y no hay eventos pendientes. Lambda congela el entorno de ejecución hasta que se recibe otro evento.

En la fase "**shut down**", Lambda termina el entorno de ejecución después de un período de inactividad o cuando se alcanza el límite máximo de invocaciones para una instancia

del entorno de ejecución. Esta fase finaliza cuando el tiempo de ejecución y todas las extensiones se han completado y no hay eventos pendientes.

Además hay una serie de cosas a tener en cuenta en este ciclo de vida, que pueden hacer que se alargue, aumentando tiempo y coste, por ejemplo:

- **Tiempo de arranque**: Conocido como "cold start", existe un ligero tiempo de arranque inicial cuando se invoca una función Lambda debido a la necesidad de aprovisionar y configurar los recursos necesarios.

- **Límites de tiempo de ejecución**: AWS Lambda impone límites de tiempo de ejecución predefinidos. Si necesitas una función que dure varias horas, no podrás hacerla en Lambda, ya que si el código excede este límite, la ejecución se interrumpe y se devuelve un error.

- **Gestión de dependencias**: El manejo de dependencias externas en una función Lambda puede ser complicado, ya que se deben incluir en el paquete de implementación. Esto puede aumentar el tamaño de la función y dificultar su administración.

Ejemplo de función Lambda

Se han creado varias funciones Lambda, una para cada uno de los endpoint de la API Gateway que hemos visto anteriormente veamos un ejemplo de una de las funciones lambdas implementadas, en este caso la encargada de añadir una receta a la base de datos, **addRecipeItem**.

```
// default imports
const AWSXRay = require("aws-xray-sdk-core");
const AWS = AWSXRay.captureAWS(require("aws-sdk"));
const { metricScope, Unit } = require("aws-embedded-metrics");
const DDB = new AWS.DynamoDB({ apiVersion: "2012-10-08" });
const { v1: uuidv1 } = require("uuid");

// environment variables
const { TABLE_NAME, ENDPOINT_OVERRIDE, REGION } = process.env;
const options = { region: REGION };
AWS.config.update({ region: REGION });

if (ENDPOINT_OVERRIDE !== "") {
  options.endpoint = ENDPOINT_OVERRIDE;
}

const docClient = new AWS.DynamoDB.DocumentClient(options);

// response helper
const response = (statusCode, body, additionalHeaders) => ({
  statusCode,
  body: JSON.stringify(body),
  headers: {
    "Content-Type": "application/json",
    "Access-Control-Allow-Origin": "*",
    ...additionalHeaders,
```

```
  },
});

function isValidRequest(event) {
  console.log("event in isValidRequest add recipe: " + event)
  return event.body !== null;
}

function getCognitoUsername(event) {
  let authHeader = event.requestContext.authorizer;
  if (authHeader !== null) {
    return authHeader.claims["cognito:username"];
  }
  return null;
}

function addRecord(event) {
  let usernameField = {
    "cognito-username": getCognitoUsername(event),
  };

  // auto generated date fields
  let dISO = new Date().toISOString();
  let auto_fields = {
    id: uuidv1(),
    creation_date: dISO,
    lastupdate_date: dISO,
  };

  //merge the json objects
  let item_body = {
    ...usernameField,
    ...auto_fields,
    ...JSON.parse(event.body),
  };

  console.log(item_body);

  //final params to DynamoDB
  const params = {
    TableName: TABLE_NAME,
    Item: item_body,
  };

  return docClient.put(params);
}

// Lambda Handler
exports.addRecipeItem = metricScope((metrics) => async (event,
   context) => {
  metrics.setNamespace("RecipeApp");
  metrics.putDimensions({ Service: "addRecipe" });
  metrics.setProperty("RequestId", context.requestId);

  if (!isValidRequest(event)) {
```

```
    metrics.putMetric("Error", 1, Unit.Count);
    return response(400, { message: "Error: Invalid request" });
  }

  try {
    let data = await addRecord(event).promise();
    metrics.putMetric("Success", 1, Unit.Count);
    return response(200, data);
  } catch (err) {
    metrics.putMetric("Error", 1, Unit.Count);
    return response(400, { message: err.message });
  }
});
```

Veamos paso por paso que es lo que hace el código.

1. Importación de módulos y configuración:

   - Se importan varios módulos de AWS necesarios para interactuar con servicios como AWS X-Ray, DynamoDB y Cognito.
   - Se obtienen algunas variables de entorno (environment variables) definidas en la configuración del entorno Lambda, como el nombre de la tabla de DynamoDB a utilizar, la región de AWS y una posible variable de ajuste del endpoint (ENDPOINT_OVERRIDE).
   - Luego, se crea una instancia de la clase DocumentClient de DynamoDB para facilitar las operaciones con la base de datos.

2. Funciones auxiliares:

   - `isValidRequest(event)`: Esta función verifica si la solicitud (event) recibida no es nula o vacía, lo que significa que se ha enviado una solicitud válida para agregar una receta.
   - `getCognitoUsername(event)`: Esta función extrae el nombre de usuario del contexto de autorización de Cognito, que proporciona información sobre el usuario que realiza la solicitud.

3. Función principal `addRecord(event)`:

   - Esta función toma la solicitud (`event`) que contiene la información de la receta que se va a agregar.
   - Primero, extrae el nombre de usuario del contexto de autorización de Cognito utilizando la función `getCognitoUsername(event)`.

- Luego, genera campos adicionales, como un ID único generado automáticamente, y las fechas de creación y última actualización, para completar el registro de la receta.

- Después, combina todos estos campos en un objeto JSON llamado `item_body`, que representa el registro completo a ser agregado a la tabla de DynamoDB.

- Finalmente, se utilizan los parámetros definidos en el objeto `params` para realizar una operación de escritura (`put`) en la tabla de DynamoDB, utilizando la instancia de DocumentClient previamente creada.

4. Lambda Handler:

   - La función Lambda principal se denomina `addRecipeItem`.

   - Utiliza el módulo `aws-embedded-metrics` para recolectar métricas relacionadas con el rendimiento de la función y el estado de las operaciones.

   - Comprueba si la solicitud recibida es válida utilizando `isValidRequest(event)`. Si no es válida, se registra un error y se devuelve una respuesta de error con un código de estado 400 (Bad Request).

   - Si la solicitud es válida, se intenta agregar el registro a la tabla de DynamoDB mediante la función `addRecord(event)`.

   - Si la operación de escritura es exitosa, se registra un éxito en las métricas y se devuelve una respuesta con un código de estado 200 (OK) y los datos del registro agregado.

   - Si ocurre un error en el proceso, se registra un error en las métricas y se devuelve una respuesta de error con un código de estado 400 (Bad Request) y un mensaje de error detallado.

El proceso es bastante similar en todas las lambdas, dónde las buenas prácticas consisten en tener la funcionalidad del código divido en diferentes funciones fuera del Handler y llamar a estos métodos desde él, para poder abstraerse de los detalles de implementación de AWS y ser un código más limpio, reusable y que se pueda exportar con mayor facilidad.

5.1.5 Amazon DynamoDB

Teniendo ya las funciones Lambda, vemos que se va a hacer uso de la base de datos DyanamoDB.

Como ya hemos explicado brevemente, Amazon DynamoDB es un servicio de base de datos **NoSQL** completamente administrado y altamente escalable que forma parte de los servicios de AWS.

DynamoDB se basa en el modelo de datos de **clave-valor**, lo que significa que los datos se almacenan y recuperan utilizando una clave única. El diseño del esquema de la base de datos se basa en **tablas**. Cada tabla está compuesta por una o más **particiones**, y cada partición contiene un conjunto de **ítems**. Los ítems, a su vez, están formados por **atributos**. Un esquema de esto se puede ver en las figuras 5.2 y 5.3

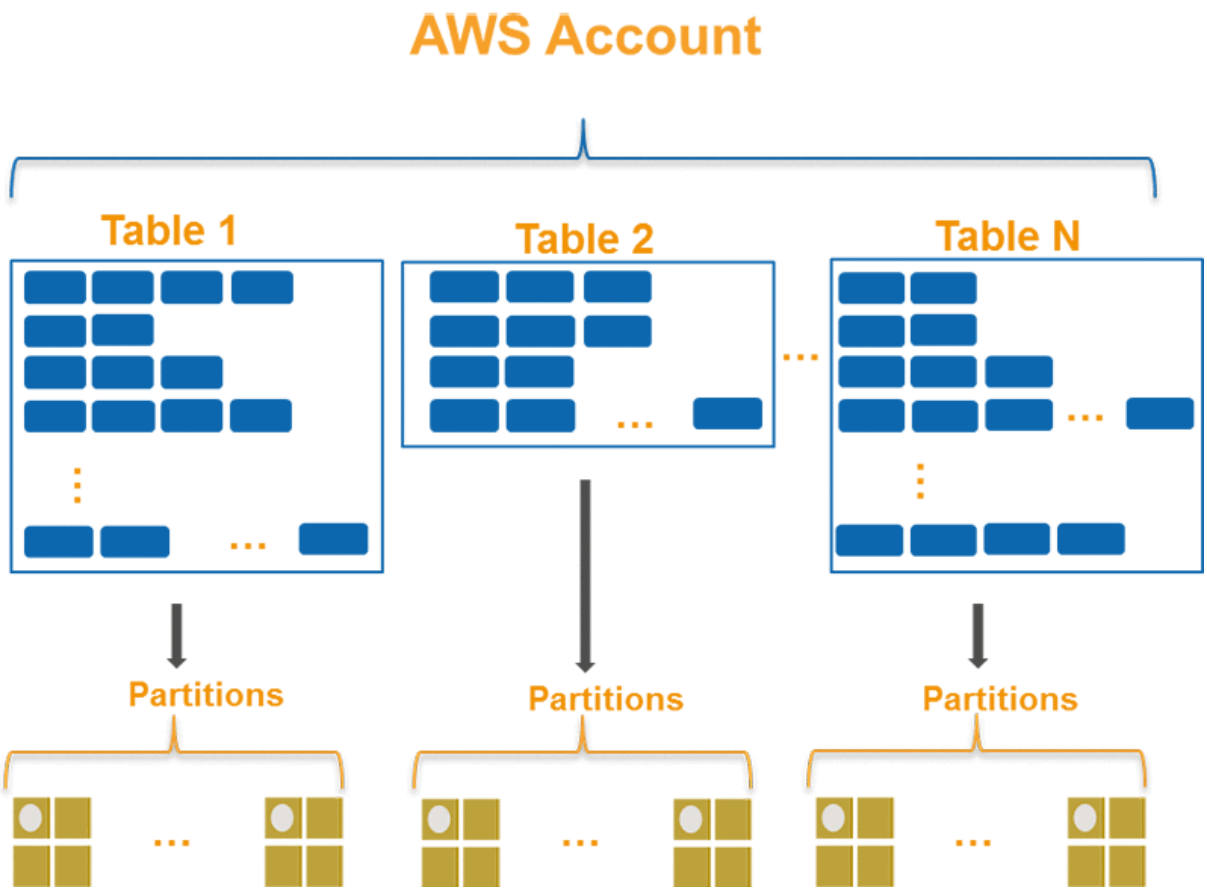


Figura 5.2: Esquema de DynamoDB - Tablas y particiones

Cuando se crea una tabla en DynamoDB, se deben definir los atributos clave. Los **atributos clave** son necesarios para identificar y acceder a los ítems de la tabla. DynamoDB admite dos tipos de claves: claves de **partición únicas** y claves de **partición-compuesta**.

Los atributos no clave se utilizan para almacenar datos adicionales relacionados con los ítems. Es importante considerar cuidadosamente los atributos clave al diseñar el esquema de la base de datos para garantizar un acceso eficiente a los datos.

En nuestro diseño, vamos a utilizar clave de partición compuesta, donde la primaria o de partición será el hash del nombre de usuario de Cognito, y la secundaria o de ordenación será el id de esa receta.

La estructura con AWS SAM para esto sería:

```
RecipeTable:
    Type: AWS::DynamoDB::Table
    Properties:
```

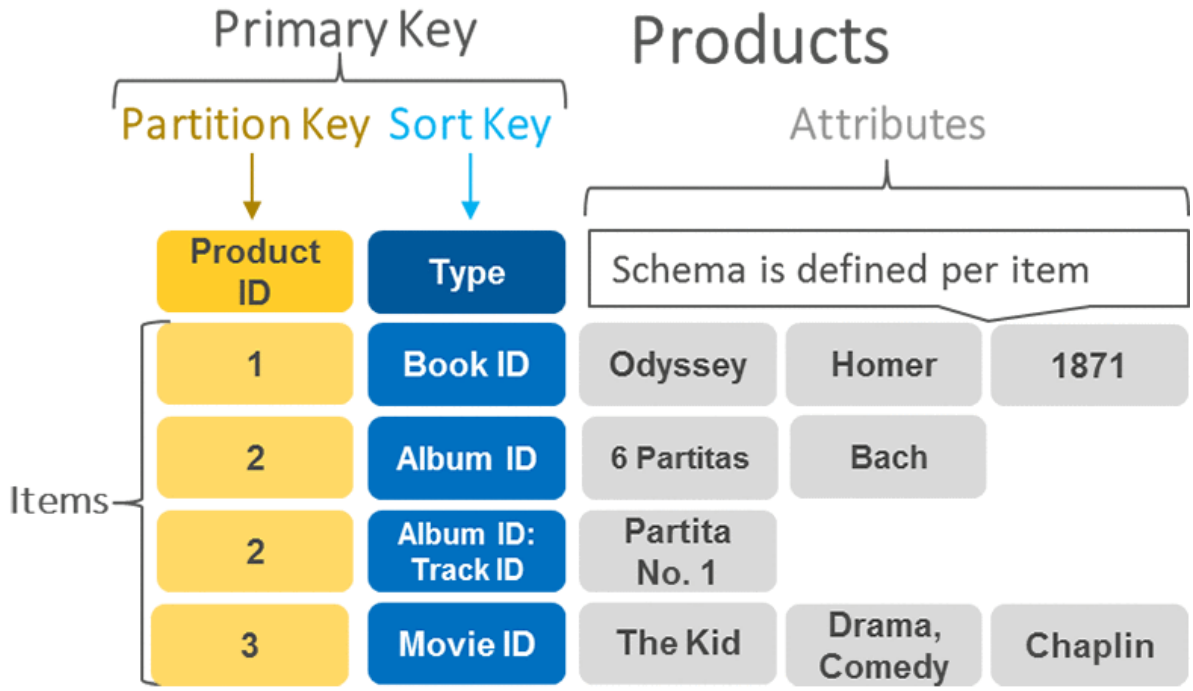


Figura 5.3: Esquema de DynamoDB - Tablas y particiones

```
TableName: recipe-table
KeySchema:
    - AttributeName: cognito-username
      KeyType: HASH
    - AttributeName: id
      KeyType: RANGE
AttributeDefinitions:
    - AttributeName: cognito-username
      AttributeType: S
    - AttributeName: id
      AttributeType: S
ProvisionedThroughput:
    ReadCapacityUnits: 5
    WriteCapacityUnits: 5
SSESpecification:
  SSEEnabled: true
```

Y al ser esta la tabla de las recetas, los atributos serán: nombre de la receta, si la receta está marcada como favorita o no y descripción de la receta. Además de unos campos extra como fecha de creación y fecha de última modificación.

Pero hay que tener en cuenta, que al ser esta una base de datos no relacional, estos atributos pueden cambiar fácilmente e incluso ser distintos para cada Item.

5.1.6 AWS Amplify

Hasta ahora hemos visto el Backend de nuestra arquitectura, que se encarga de la lógica y el almacenamiento de datos. En la siguiente sección veremos cómo hemos creado el frontend, que es la interfaz gráfica que el usuario ve en su navegador. Pero para **servir** esta **interfaz** al cliente, usaremos AWS Amplify Hosting, un servicio que nos permite **alojar** y **distribuir** nuestra página web de forma fácil y segura.

Amplify es un servicio de alojamiento web estático y escalable que forma parte de los servicios de AWS. Amplify Hosting permite implementar y servir aplicaciones web estáticas, como sitios web o aplicaciones de página única (SPA), directamente desde AWS.

Otra alternativa para esto sería usar **Amazon S3**, pero AWS Amplify Hosting ofrece ventajas en comparación con el uso directo de Amazon S3 para alojar aplicaciones web estáticas. Amplify proporciona una capa de abstracción que simplifica la configuración y el despliegue, integra servicios adicionales de Amplify, automatiza el proceso de CI/CD, ofrece funcionalidades específicas de hosting, como redireccionamientos y compresión de archivos, y garantiza escalabilidad y rendimiento con la CDN de Amazon CloudFront. Amplify también se integra con Amplify Console para una gestión más sencilla.

En nuestro caso, vamos a utilizar un repositorio de código fuente (GitHub) para facilitar el seguimiento de los cambios y la implementación automática cuando se realizan actualizaciones en el repositorio. Esto es lo que se conoce como **CI/CD**, integración y despliegue continuos. Veamos como se implementa esto en la estructura de AWS SAM.

```
AmplifyApp:
    Type: "AWS::Amplify::App"
    Properties:
      Name: RecipeApp
      Repository: !Ref Repository
      Description: Recipe app
      OauthToken: !Ref OauthToken
      BuildSpec: |-
        version: 0.1
        frontend:
          phases:
            build:
              commands:
                - cd www/src
                - npm install
                - npm run build
          artifacts:
            baseDirectory: www/build/
            files:
              - '**/*'
      Tags:
        - Key: Name
          Value: Recipe
      IAMServiceRole: !GetAtt AmplifyRole.Arn
```

Una cosa importante a tener en cuenta, es que nuestra página web será lo que se conoce como **Single-Page-Application**, es decir, una aplicación web que se carga como una sola página en el navegador y ofrece una experiencia interactiva y fluida al usuario. En lugar de cargar páginas completas cada vez que se realiza una acción, una SPA carga y actualiza dinámicamente el contenido en la misma página, utilizando tecnologías como JavaScript y AJAX.

Por lo que si queremos usar **rutas**, para simular esta navegación y organizar mejor el código y la estructura de la página, tenemos que hacer que se redireccionen al /index.html, para hacer esto se haría con:

```
CustomRules:
    - Source: '</^[^.]+$|\.(?!(css|gif|ico|jpg|js|png|txt|svg|
       woff|woff2|ttf|map|json|webp)$)([^.]+$)/>'
      Target: "/index.html"
      Status: "200"
```

5.1.7 GitHub

GitHub es una plataforma de desarrollo colaborativo que permite alojar proyectos usando el sistema de control de versiones **Git**.

Se creará un repositorio en GitHub para almacenar el código del proyecto. Dado que se trata de un proyecto individual, no será necesario establecer una política de colaboradores. Sin embargo, se utilizará GitHub como una herramienta para tener un control de versiones y permitir la integración de despliegue continuo (**CI/CD**) con AWS Amplify. Esto facilitará el seguimiento de los cambios realizados en el código y permitirá una implementación más eficiente.

5.1.8 OpenAI API

La API de OpenAI ofrecerá la capacidad de generar las recetas personalizadas, para esto usaremos el modelo **GPT-3.5 Turbo**. Este modelo de inteligencia artificial es uno de los más avanzados disponibles actualmente, además de ser bastante eficiente en cuanto a rapidez y coste.

Para generar estas recetas, se enviará el texto creado a partir de una plantilla que se rellenara con la información proporcionada por el usuario. Ejemplos de estas plantillas son:

- Crea una receta teniendo en cuenta lo siguiente: Ingredientes disponibles: /ingredientes/. Herramientas de cocina disponibles: /herramientas/. Tiempo disponible: /tiempo/. Dificultad de la receta: /dificultad/.

- Créame una receta /restricción alimenticia/ que tenga los siguientes macros: /proteína/ gramos de proteína, /carbohidratos/ gramos de carbohidratos, /grasas/ gramos de grasa.

- Créame una receta /restricción alimenticia/ para /comida/cena../ estilo /asiático/-mejicano../

Esta API se puede llamar fácilmente desde una función Lambda teniendo en cuenta la necesidad de importar el paquete de OpenAI de la siguiente manera:

```
const completion = await openai.createChatCompletion({
  model: "gpt-3.5-turbo",
  messages: [{role: "user", content: body.question}],
  max_tokens: 2048
});

return response(200, { body:  completion.data.choices[0].
   message.content });
```

Esta respuesta se recogerá en la web de forma asíncrona y se mostrará al usuario, pero hay que tener en cuenta que la generación de la receta no es un proceso instantáneo, tarda unos segundos, por lo que darle información al usuario de que su receta se está generando es importante, esto se ha hecho bloqueando el botón de generar receta y aplicándole la animación de un símbolo de carga (Spinner).

5.2 Desarrollo de la web

5.2.1 Interfaz

El diseño de la página web se centrará en ser sencillo pero atractivo visualmente, con el objetivo de ofrecer una experiencia de usuario agradable y atractiva. Se utilizarán elementos visuales y colores cuidadosamente seleccionados para crear un diseño atractivo y agradable a la vista. La simplicidad del diseño permitirá que los usuarios se familiaricen rápidamente con la interfaz y naveguen sin problemas por la página web.

Formularios pre-rellenados con información común

Con el objetivo de agilizar el proceso de creación de recetas, se implementarán formularios pre-rellenados con las herramientas que suelen estar en todas las cocinas o otras cosas comunes. Esto permitirá que los usuarios ingresen datos específicos de una receta, de manera más rápida y conveniente. Una posible implementación se ve en la figura 5.4

Guardado y gestión de recetas

Se permitirá a los usuarios guardar y gestionar sus recetas creadas. Después de crear una receta, los usuarios podrán guardarla. Esto les brindará la posibilidad de acceder fácilmente a sus recetas guardadas en el futuro y gestionarlas. Se implementarán funciones intuitivas y fáciles de usar para garantizar una gestión eficiente de las recetas. Una posible implementación se ve en la figura 5.5.

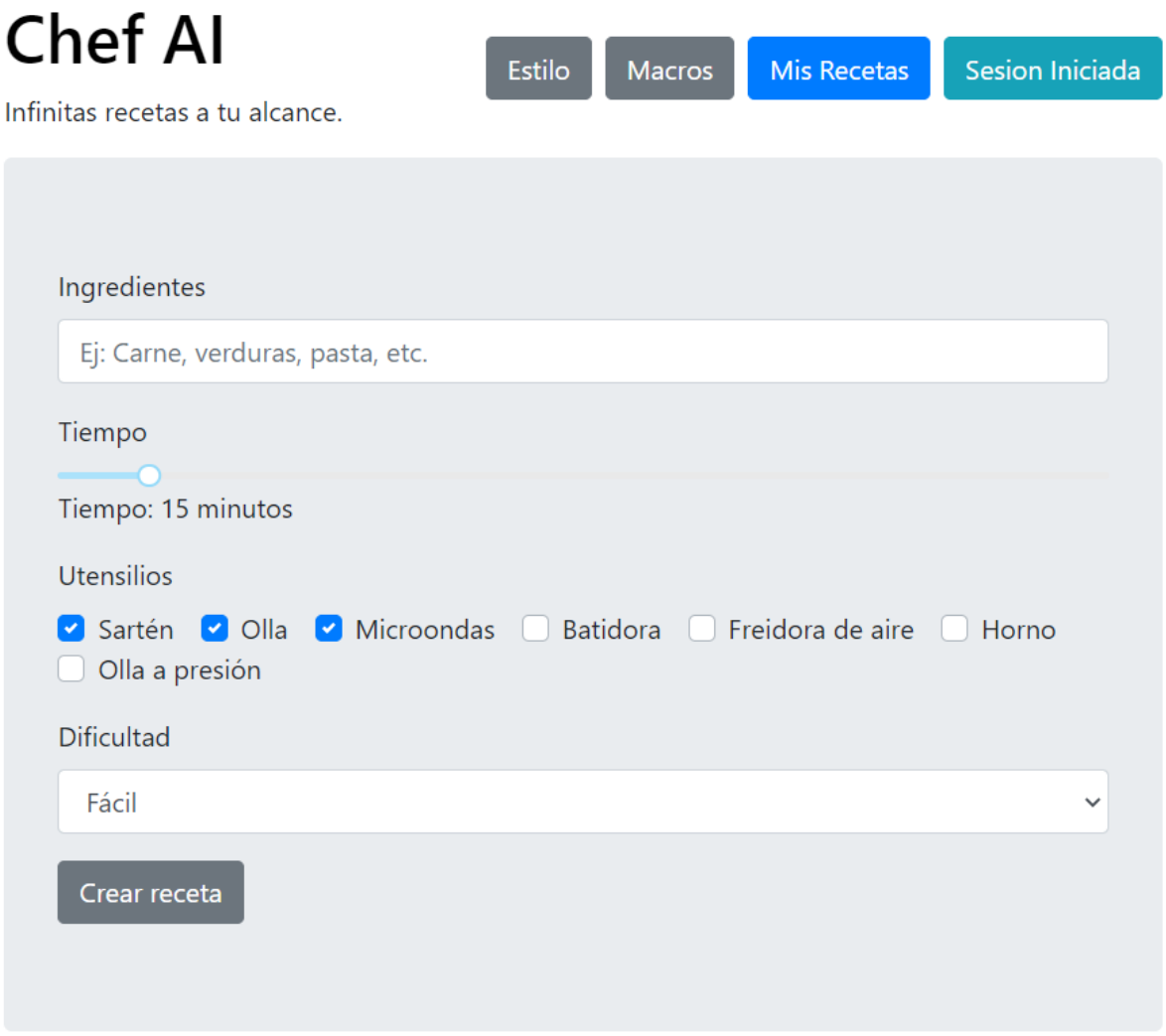


Figura 5.4: Mockup Diseño interfaz - Creación de recetas

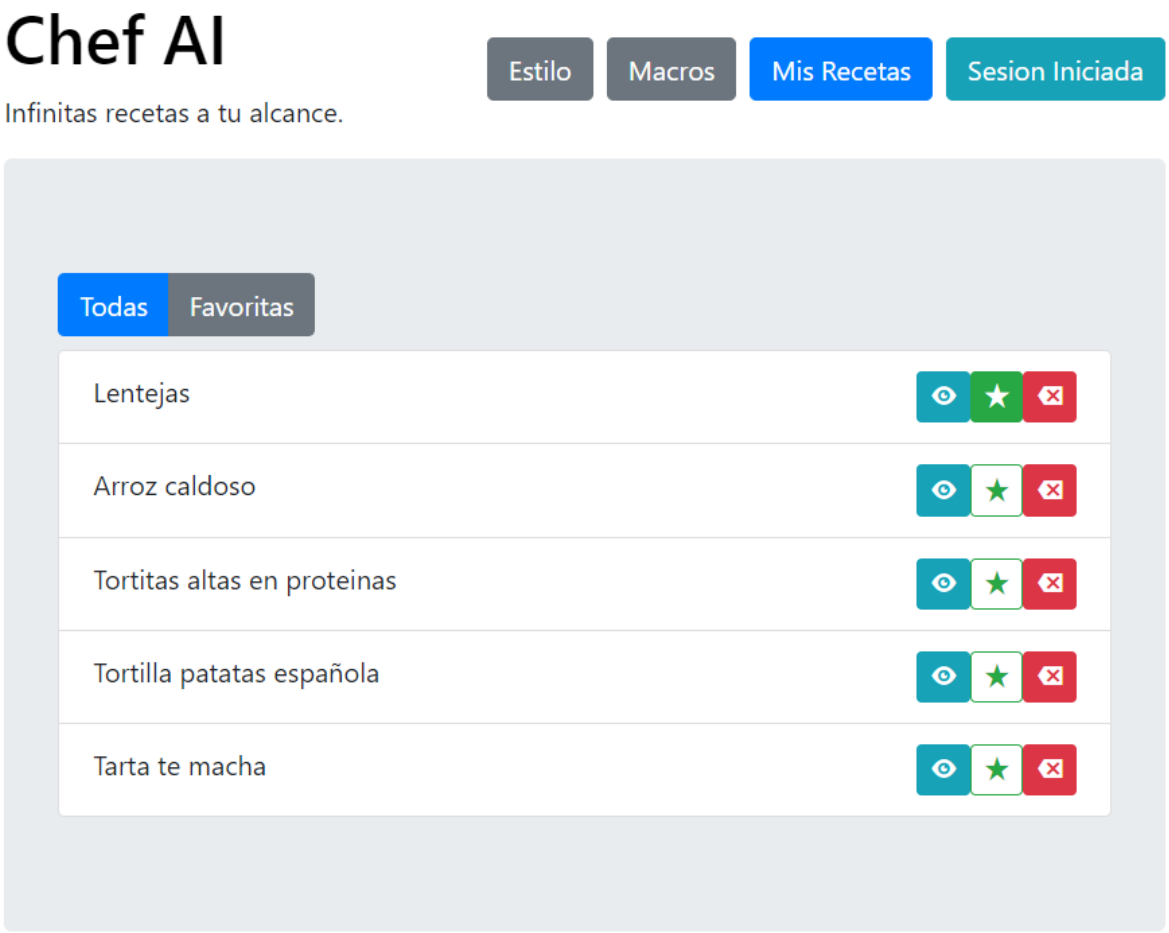


Figura 5.5: Mockup Diseño interfaz - Listado de recetas

5.2.2 Implementación

React

Se ha utilizado React para implementar esta SPA (Single Page Application). React se basa en un enfoque de desarrollo basado en **componentes**, lo que significa que la interfaz se divide en componentes reutilizables que se actualizan de forma eficiente cuando los datos cambian.

Una de las principales ventajas de utilizar React es el uso de **JSX**, una extensión de JavaScript que permite escribir código similar a HTML dentro de los componentes. JSX facilita la mezcla de la lógica JavaScript y la estructura HTML de manera intuitiva. Esto permite mostrar elementos o componentes de manera más sencilla según el estado de la aplicación, mostrar texto dinámico basado en propiedades y mucho más.

React ofrece varias herramientas y patrones para implementar estas funcionalidades:

Estados y propiedades: React utiliza el concepto de estados (state) y propiedades (props) para manejar y pasar datos entre componentes. Los estados permiten que los componentes se actualicen y reflejen los cambios en los datos de manera eficiente. Los estados se utilizan para almacenar y manipular datos que pueden cambiar a lo largo del tiempo, como la información de un formulario o el estado de autenticación de un usuario.

Eventos y manejo de interacciones: El manejo de eventos en React es fundamental para capturar las acciones del usuario, como clics de botones o cambios en formularios. React proporciona una sintaxis sencilla para asociar funciones de controlador de eventos a elementos de la interfaz. Estas funciones se ejecutan cuando ocurre el evento y permiten actualizar el estado de la aplicación en respuesta a las interacciones del usuario.

Estas características de React, como el uso de componentes reutilizables, JSX para una sintaxis más clara y concisa, y las herramientas para manejar estados y eventos, hacen que el desarrollo de una SPA sea más eficiente y escalable. React simplifica el proceso de crear interfaces de usuario interactivas y dinámicas, lo que resulta en una experiencia de usuario más fluida y atractiva.

Bootstrap

Además de utilizar React, se empleará **Reactstrap**, una biblioteca que facilita la integración de Bootstrap en proyectos de React. Reactstrap proporciona **componentes preestilizados** de Bootstrap, que van de elementos simples como: botones, listas o formularios hasta cosas más complejas como carruseles, barras de navegación, etc. listos para ser utilizados en una aplicación React, lo que agiliza el desarrollo y permite crear interfaces atractivas y receptivas de manera más eficiente.

Una de las principales ventajas de Reactstrap es la disponibilidad de una amplia gama de componentes de interfaz de usuario (UI) altamente personalizables. Estos componentes incluyen elementos como botones, barras de navegación, formularios, tarjetas y muchos otros que siguen las directrices y el diseño responsivo de Bootstrap.

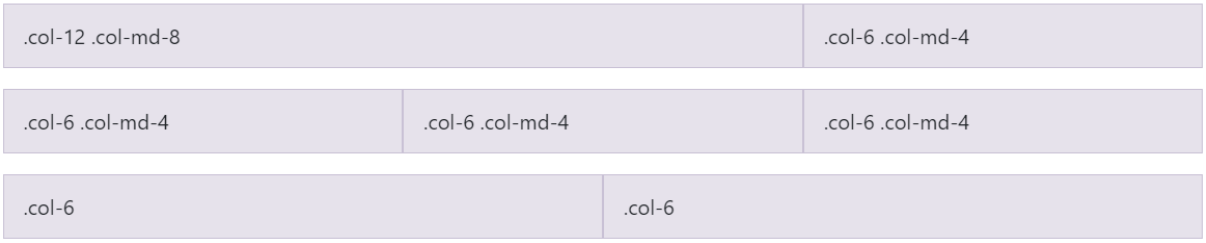


Figura 5.6: Ejemplo grid de Bootstrap

Y además de los componentes, se puede usar el sistema **grid** de Bootstrap lo que facilita la creación de **diseños responsivos** y adaptativos. Con este grid, se puede organizar y distribuir los componentes en diferentes tamaños de pantalla de manera fluida y coherente. Se puede ver en la figura 5.6 un ejemplo de este.

Al utilizar Reactstrap junto con React, se pueden aprovechar las características y beneficios de ambas bibliotecas. Reactstrap simplifica el proceso de implementación de componentes de Bootstrap en una aplicación React, lo que ahorra tiempo y esfuerzo en el desarrollo de la interfaz de usuario. Además, al seguir las directrices de diseño de Bootstrap, se garantiza una apariencia coherente y profesional en toda la aplicación.

Ejemplo de un componente

Teniendo todo esto en cuenta, podemos crear nuestros propios componentes, permitiendo reutilizar código y mantener el comportamiento en diferentes partes de la aplicación, por ejemplo, uno de los componentes desarrollados es el componente "**Recipe**", que será el encargado de formatear el texto devuelto por la API de OpenAI, y mostrar si quieres guardar o no la receta, la implementación de este componente es la siguiente:

```
import { useState } from 'react';
import { Button, Form, FormGroup, Input } from 'reactstrap';
import ShowRecipe from './ShowRecipe'

function Recipe({ recipeText, addRecipe }) {

  const [buttonText, setButtonText] = useState("Guardar");
  const [buttonDisabled, setButtonDisabled] = useState(false);

  function addRecipeClick(recipeText) {
    setButtonText("Guardado");
    setButtonDisabled(true);
    addRecipe(recipeText);
  }

  return (
    <div className="Recipe">

      <ShowRecipe recipeText={recipeText} />

      <h2>Si te ha gustado la receta guardala!</h2>
      <Form className="d-flex" inline>
        <FormGroup className="flex-fill">
```

```
            <Input
              type="text"
              name="recipe"
              id="newRecipe"
              placeholder="nombre de la receta"
              className="w-100"
            />
          </FormGroup>
          <Button
            onClick={(e) => addRecipeClick(recipeText)}
            color="primary"
            className="ml-1"
            disabled={buttonDisabled}
          >
            {buttonText}
          </Button>
        </Form>
      </div >
    );
}

export default Recipe;
```

Este componente se podrá reutilizar en otros componentes simplemente utilizando su etiqueta y pasándole como parámetro el texto a mostrar y la función que tiene que llamar para guardar la receta.

```
<Recipe
  recipeText={recipeText}
  addRecipe={addRecipe}
/>
```

6 Implantación

Este capítulo tiene como objetivo brindar una visión general y guía práctica sobre el despliegue de aplicaciones. Comenzaremos con una introducción a los conceptos clave y la importancia del despliegue adecuado de aplicaciones. Luego, nos adentraremos en el proceso detallado para desplegar una aplicación utilizando AWS SAM.

6.1 Pasos previos

Para hacer uso de la aplicación y llevar a cabo su despliegue, se deben seguir una serie de pasos, de forma correcta, para que los distintos servicios se puedan comunicar entre ellos.

Como se ha visto en la seccion 5.1.1, AWS SAM (Serverless Application Model) es la herramienta que nos va a ayudar a implementar los servicios, pero hay que tener en cuenta varias cosas antes de empezar el despliegue.

Stack

Cuando creemos estos servicios, vamos a crearlos todos bajo el mismo "STACK_NAME", esto es una colección de recursos para que manejarlos sea más ordenado y sencillo, permitiendo que se traten todos los recursos de la aplicación como uno solo, haciendo esto posible cosas como por ejemplo: borrar todos los servicios a la vez, aplicar **rollbacks** en caso de que un paso en el despliegue falle, filtrar servicios por stack, etc.

Región

En AWS, cuando creamos servicios, debemos especificar una región. Una región en AWS es una ubicación geográfica donde Amazon Web Services tiene sus centros de datos. Se puede observar en 6.1, que hay centros de datos en España, Irlanda, Londres, París,etc.

Cada región es autónoma y está compuesta por varias zonas de disponibilidad. Al seleccionar una región, debemos considerar factores como la latencia, el rendimiento y el cumplimiento normativo. Algunos servicios son globales y se pueden acceder desde cualquier región. Otros solo están disponibles en determinadas regiones. Es importante elegir la región adecuada para nuestras necesidades.

Figura 6.1: Regiones de AWS en Europa

En nuestro caso, utilizaremos **Irlanda**, también conocido como **eu-west-1**, por varias razones siendo la principal que es una de las regiones más antiguas y establecidas de AWS. Como resultado, ofrece una amplia gama de servicios y características como buen precio y latencia.

Creación de cuentas y tokens

Para hacer uso de los servicios tanto de Amazon, como de OpenAI y GitHub, se necesitan crear las respectivas cuentas de usuario y gestionarlas, añadiendo permisos, repositorios y disponer de los fondos suficientes en el caso de ser necesario. Para el desarrollo de este proyecto se han podido utilizar estas cuentas con el conocido periodo de prueba, pero en caso de querer realizar un lanzamiento al mercado, habría que tener en cuenta estos fondos.

Una vez se hayan creado las cuentas necesarias, es importante obtener las credenciales de acceso remoto correspondientes, como tokens, claves de API u otros identificadores, para poder manejar las cuentas desde un entorno remoto o a través de la aplicación desarrollada.

6.2 Despliegue

Una vez hemos realizado la preparación del entorno y cuentas, tenemos la estructura del proyecto en el template.yaml de forma correcta y el código desarrollado. El despliegue consiste en cinco pasos:

1. **Empaquetado y compilación de la aplicación**: Utilizar el comando sam build para construir la aplicación y sus dependencias a partir del fichero template.yaml. Este comando analizará el código fuente, identificará las dependencias necesarias y las empaquetará en un formato compatible con AWS Lambda.

2. **Despliegue de la aplicación**: Utilizar el comando sam deploy para desplegar la aplicación. Este comando creará o actualizará los recursos definidos en la plantilla de AWS SAM en la cuenta de AWS convirtiendo la plantilla a CloudFormation.

3. **Obtención de datos de los servicios:** Después de desplegar los servicios en AWS, es necesario obtener los datos relevantes, como las URL de las APIs, el cliente del pool de usuarios y otra información necesaria para interactuar con los servicios desde tu aplicación web.

4. **Integración de los datos en el código de la aplicación web**: Una vez que hemos obtenido los datos de los servicios, integrarlos en el código de la aplicación web. Esto implica modificar las configuraciones o variables pertinentes dentro del código para que la aplicación pueda comunicarse correctamente con los servicios de AWS.

5. **Subir los cambios a GitHub y despliegue continuo**: Finalmente, cuando se han realizado las modificaciones necesarias en el código de la aplicación web, se tienen que subir los cambios al repositorio de GitHub. Esto implica utilizar el comando git push para enviar los cambios locales al repositorio remoto. Y una vez que los cambios se hayan subido correctamente se va a aprovechar la funcionalidad de despliegue continuo para que la aplicación web se implemente automáticamente.

7 Resultado

En este capítulo, se presentará un recorrido detallado de la aplicación, destacando sus diferentes componentes, características y funcionalidades. Se explorarán las diversas secciones y módulos, proporcionando una visión completa y comprensiva de la aplicación desarrollada.

7.1 Landing page

Esta es la primera página que se verá al acceder a la web, por lo que la idea es que sea sencilla y descriptiva, explique que es la web, que servicios tiene.

La idea principal de la Landing page es atraer la atención de los visitantes y motivarlos a explorar más a fondo la web. Para lograr esto, se ha utilizado un diseño visualmente atractivo y se ha destacado la usabilidad de la plataforma.

Además cuando un usuario no esté registrado y autenticado, esta será la única página que podrá ver.

Se puede observar en la figuras 7.1 y 7.2 el resultado, véase como la página cambia al estar el usuario autenticado o no, ofreciendo enlaces a las secciones nombradas para el usuario con permisos y un aviso de que no está con la sesión iniciada al usuario anónimo, además del cambio en la barra de navegación.

7.2 Creación de recetas

A la hora de crear recetas, se han creado 3 servicios que solventan distintos propósitos. A la hora de crear recetas, se han desarrollado tres servicios distintos para cumplir diferentes propósitos. A continuación, se ve cada uno de estos servicios.

Figura 7.1: Landing page cuando el usuario no está autenticado

7.2.1 Estilo

En el apartado de Estilo, se le brinda al usuario la capacidad de crear recetas según sus preferencias culinarias específicas. Aquí, el enfoque principal es permitirles seleccionar un estilo o categoría de recetas que se ajuste a sus gustos, como por ejemplo, un postre asiático, vegano, entre otros.

Se puede observar en la figura 7.3 el resultado.

Figura 7.2: Landing page con usuario autenticado

7.2.2 Macro

El apartado de Macro proporciona una función especializada en la creación de recetas basadas en los macronutrientes seleccionados por el usuario. Esta característica es especialmente útil para aquellos que desean seguir una dieta específica o tener un control más preciso de su ingesta de macronutrientes.

Figura 7.3: Apartado Estilo de la creación de recetas

Se puede observar en la figura 7.4 el resultado.

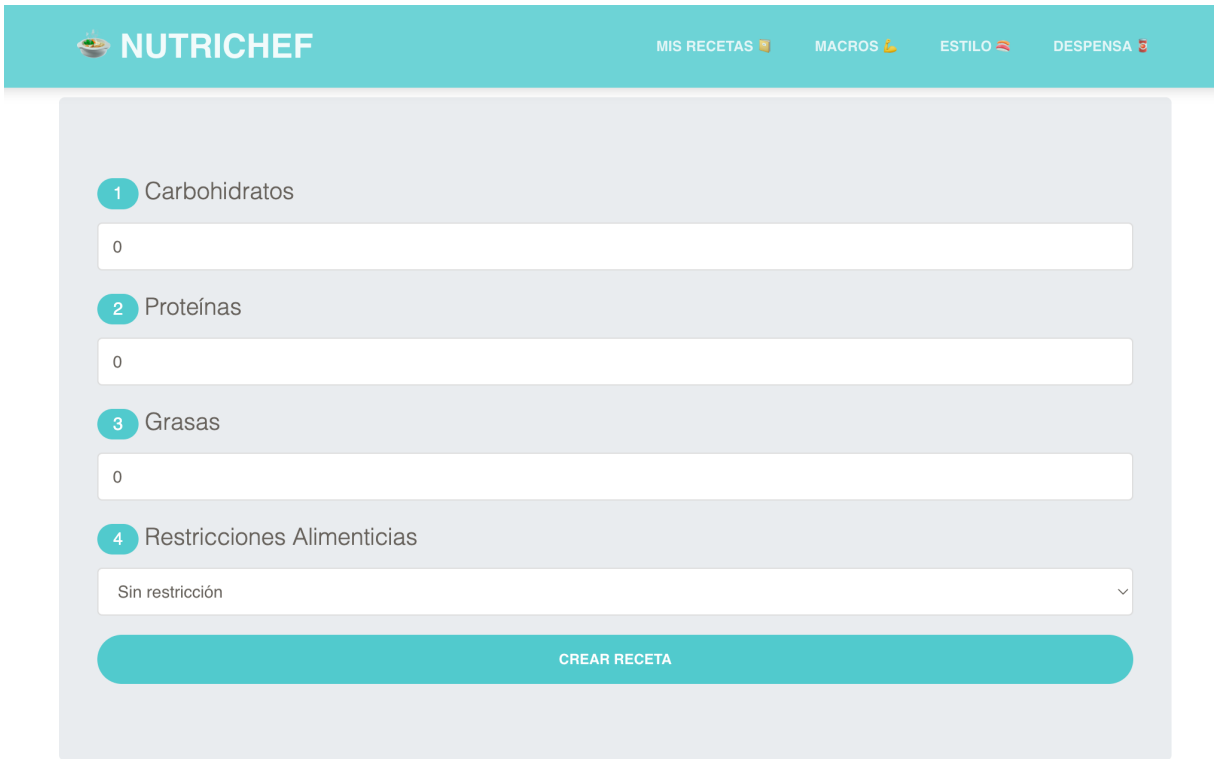


Figura 7.4: Apartado Macro de la creación de recetas

7.2.3 *Despensa*

El apartado de Despensa es una función clave de la web, ofrece a los usuarios la posibilidad de indicar los ingredientes disponibles en su cocina, tiempo, utensilios, etc. y, a partir de esa información, generar recetas que utilicen esos ingredientes. Esto facilita la planificación de comidas y reduce el desperdicio de alimentos, ya que se pueden aprovechar los ingredientes disponibles antes de comprar nuevos.

Se puede observar en la figura 7.5 el resultado.

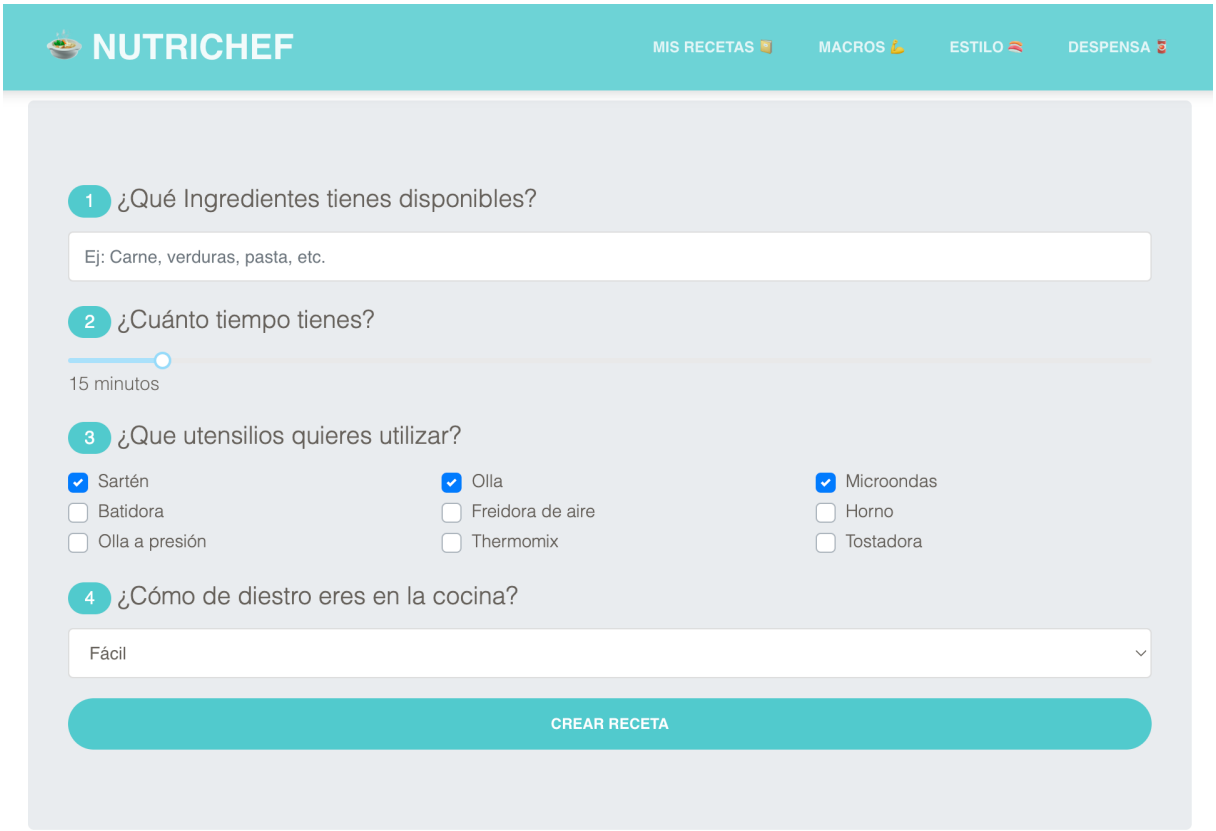


Figura 7.5: Apartado Despensa de la creación de recetas

7.2.4 *Generar la receta*

Cuando tenemos ya el formulario relleno con las opciones que queremos, al darle a generar receta, se deshabilitará el botón de generar receta y se informará al usuario de que se está generando la misma con un spinner. Una vez se genere la recetá, se mostrará debajo de este botón, y se le dará la opción al usuario de guardar la receta con el nombre que deseé.

Esto se puede observar en la figura 7.6 el resultado.

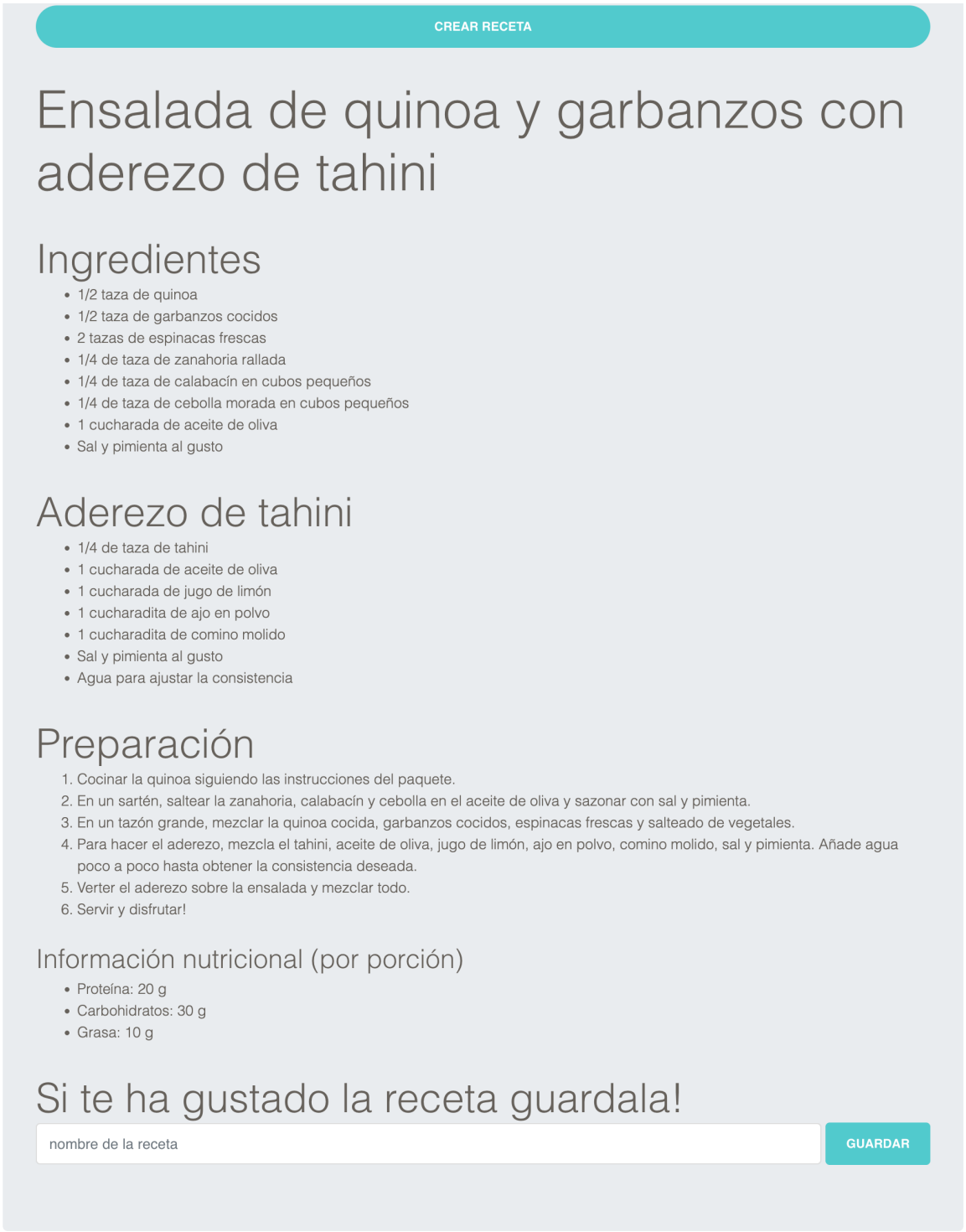


Figura 7.6: Visualización de la receta generada

7.3 Mis recetas

La página de Mis recetas permite a los usuarios almacenar y organizar sus recetas. Los usuarios tienen la posibilidad de acceder a una lista personalizada de las recetas que han guardado. Aquí pueden administrar sus recetas guardadas, marcarlas como favoritas, eliminarlas o realizar otras acciones relacionadas.

Se puede observar en la figura 7.7 el resultado.

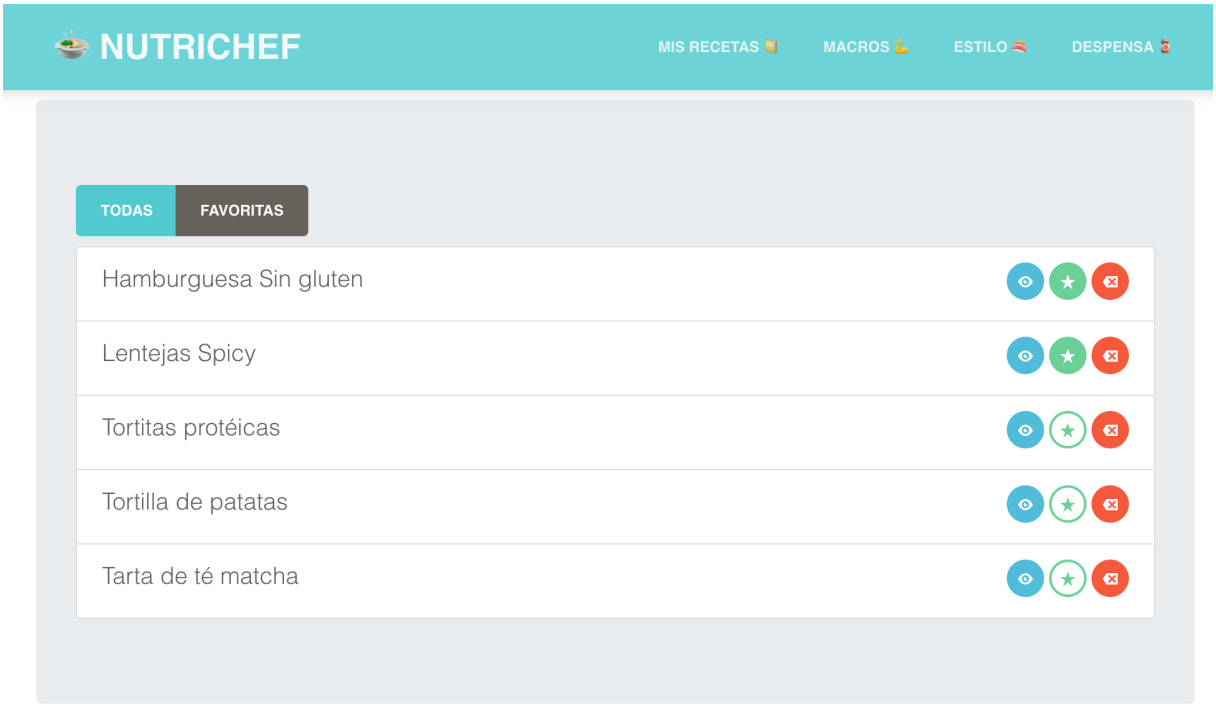


Figura 7.7: Lista de recetas guardadas

Si se quiere ver una de estas recetas de la lista, al darle al botón con un ojo, se abrirá un modal con la misma, se puede observar en la figura 7.8.

7.4 Vista en móvil

En la actualidad, el uso de dispositivos móviles para acceder a sitios web se ha vuelto cada vez más común. Por lo tanto, es crucial que una web sea responsive, es decir, que se adapte y se vea correctamente en diferentes tamaños de pantalla, incluyendo los dispositivos móviles.

Cuando una web es responsive, los elementos se reorganizan y redimensionan automáticamente para adaptarse a la pantalla del dispositivo. Esto significa que los usuarios no tendrán que hacer zoom, desplazarse horizontalmente o lidiar con una interfaz desordenada al acceder desde sus teléfonos móviles.

Como se puede apreciar en la figura 7.9, todos los elementos se ajustan de manera adecuada al tamaño de la pantalla, ofreciendo una experiencia de navegación fluida y una fácil interacción con la web.

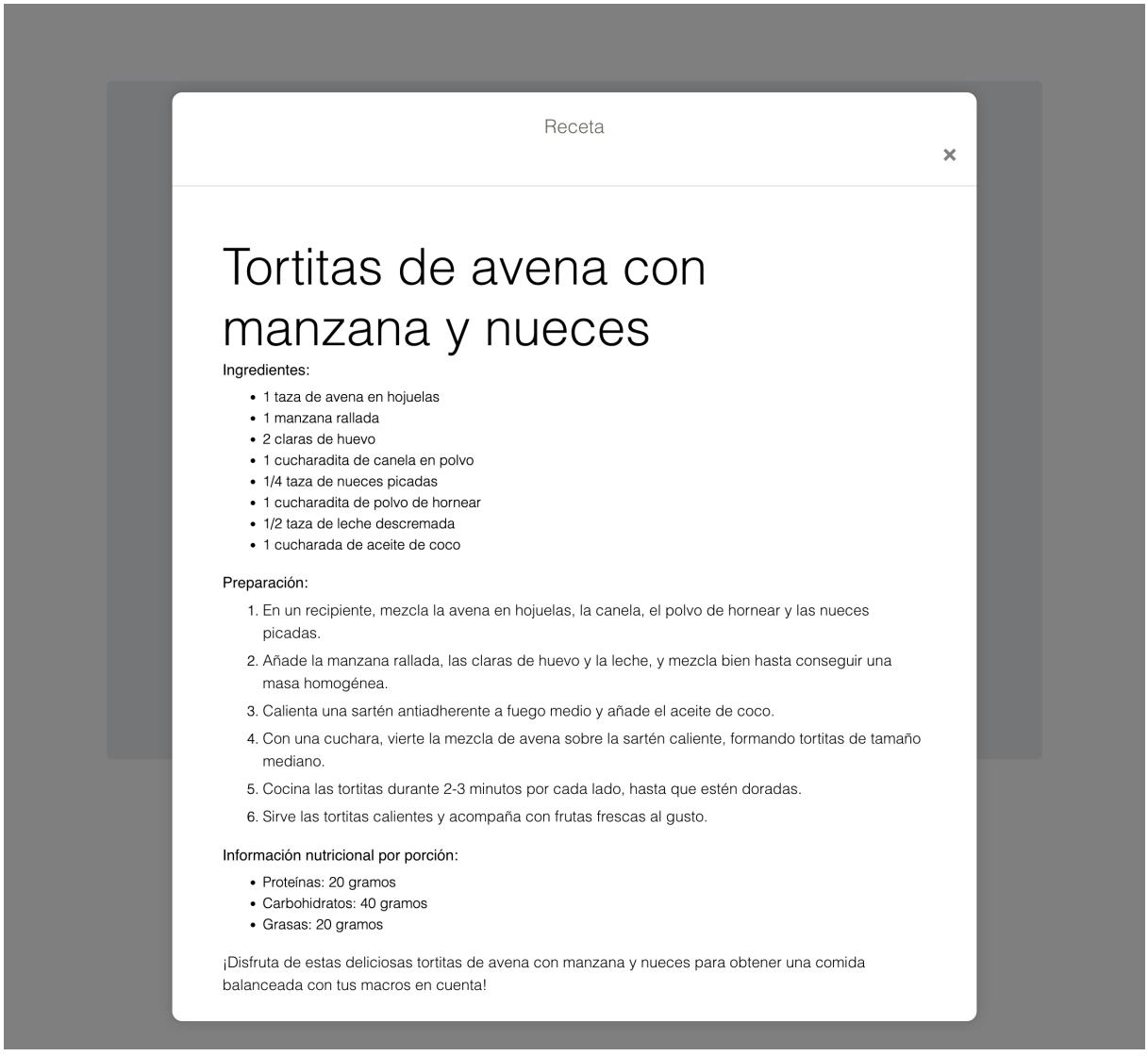


Figura 7.8: Visualización de receta guardada con modal

(a) Usuario anónimo **(b)** Página creación de recetas

Figura 7.9: Landing page vista desde móvil y navbar

8 Conclusiones

> *Las arquitecturas sin servidor ofrecen varias ventajas sobre las arquitecturas tradicionales basadas en servidor. Estas ventajas incluyen costes de desarrollo y mantenimiento reducidos, escalabilidad automática, rendimiento fiable y facilidad de uso. Sin embargo, el uso de arquitecturas sin servidor también presenta algunos retos, como el coste, la complejidad y la dependencia del proveedor. En general, las arquitecturas sin servidor son una tecnología prometedora para el desarrollo de aplicaciones web*

Una vez terminado el trabajo, es el momento de reflexionar y tratar de analizar y descubrir si éste se ha realizado correctamente y si se han cumplido los objetivos.

Para descubrir si el trabajo ha sido gestionado correctamente, así como si se han aprovechado los resultados obtenidos del mismo, hay que que fijarse en dos puntos: si se han cumplido los objetivos o no y qué conocimientos se han adquirido a lo largo de este proyecto. Estas conclusiones se basan, por lo tanto, en recuperar los objetivos formalizados del proyecto y analizarlos individualmente para descubrir si han sido alcanzados con éxito.

8.1 Cumplimiento de los objetivos

Utilización de una plataforma de computación en la nube para el despliegue de la arquitectura serverless

La utilización de una plataforma de computación en la nube, como Amazon Web Services (AWS), ha permitido desplegar la arquitectura serverless del sistema de manera eficiente. La arquitectura serverless elimina la necesidad de administrar servidores de manera manual, lo que reduce la carga de trabajo en el mantenimiento del sistema y permite centrarse en la lógica de la aplicación.

Esto ha permitido aprovechar los servicios de cómputo escalables, como AWS Lambda, Amplify, Cognito, DynamoDB, etc. Eliminando la necesidad de gestionar y escalar servidores de manera manual.

La elección de AWS como plataforma de computación en la nube se basó en su amplia gama de servicios y su confiabilidad. AWS proporciona una infraestructura escalable y

altamente disponible, lo que garantiza que el sistema pueda manejar la carga de trabajo de manera eficiente y estar disponible para los usuarios en todo momento.

Creación de la aplicación web para la generación de recetas de cocina a partir de las instrucciones del usuario

El desarrollo de la aplicación web se llevó a cabo utilizando diversas tecnologías y frameworks para garantizar una experiencia de usuario fluida y funcional. HTML, CSS y JavaScript son los lenguajes fundamentales utilizados para la construcción de la interfaz de usuario y la interacción con el sistema.

El framework React se empleó para desarrollar la aplicación web, lo que permitió crear componentes reutilizables y una interfaz de usuario interactiva. React es conocido por su rendimiento, facilidad de uso y su capacidad para actualizar partes específicas de la interfaz de usuario sin tener que recargar toda la página, lo que mejora la experiencia del usuario y reduce la carga en el servidor.

Bootstrap, otro framework utilizado, facilitó el diseño y la maquetación del sitio web. Proporciona una amplia variedad de componentes y estilos prediseñados, lo que permitió crear una interfaz atractiva y responsiva sin tener que diseñar todo desde cero.

La aplicación permite a los usuarios ingresar sus instrucciones y obtener recetas generadas en base a ellas. Se ha utilizado un enfoque de desarrollo ágil, con iteraciones y pruebas continuas para garantizar la funcionalidad y la experiencia del usuario.

Implementación de un sistema de gestión y almacenamiento de las recetas creadas por los usuarios:

Para gestionar y almacenar las recetas generadas por los usuarios, se ha utilizado Amazon DynamoDB como base de datos. DynamoDB es un servicio de base de datos NoSQL altamente escalable y administrado por AWS. Esta elección se debe a su capacidad para manejar grandes volúmenes de datos de manera eficiente y a su escalabilidad automática, lo que permite adaptarse al crecimiento del sistema sin interrupciones.

Se ha diseñado e implementado un esquema de base de datos adecuado para almacenar la información relevante de las recetas. Además DynamoDB permite un modelo de datos flexible y puede adaptarse fácilmente a los requisitos específicos de la aplicación.

La implementación de este sistema de gestión y almacenamiento de recetas permite que los usuarios guarden y accedan a sus creaciones en cualquier momento. También ofrece la posibilidad de realizar análisis de datos y obtener información valiosa sobre las preferencias y tendencias culinarias de los usuarios, lo que podría ser útil para mejorar y personalizar la experiencia de la aplicación.

Integración de una API para generar recetas a partir de las instrucciones del usuario

Para generar recetas a partir de las instrucciones ingresadas por los usuarios, se ha integrado el modelo de procesamiento del lenguaje natural de OpenAI con su API. Esta API permite la comunicación entre nuestro sistema y su modelo, que podrá analizar y comprender el texto de las instrucciones para extraer información relevante y generar recetas basadas en esa información.

Este modelo de procesamiento del lenguaje natural utiliza algoritmos y modelos de aprendizaje automático para realizar tareas como el etiquetado de partes del discurso, el análisis de entidades y la extracción de relaciones. Estos procesos permiten identificar y comprender los ingredientes, la dificultad, utensilios disponibles y otros elementos clave presentes en las instrucciones de los usuarios.

Al comprender el texto de las instrucciones, puede generar recetas que se ajusten a las preferencias y necesidades del usuario. Por ejemplo, si las instrucciones mencionan una restricción alimenticia específica, como ser vegetariano o intolerante al gluten, puede generar recetas que cumplan con esas restricciones.

La integración de esta API amplía las capacidades de la aplicación web, permitiendo que los usuarios obtengan recetas personalizadas y adaptadas a sus instrucciones. Esto facilita la generación de recetas rápidas y precisas, ahorrando tiempo a los usuarios y ofreciendo opciones culinarias acordes a sus preferencias individuales.

Obtención de una arquitectura eficiente y escalable

El enfoque de arquitectura sin servidor con el uso de plataformas en la nube como AWS, ha permitido lograr una arquitectura altamente eficiente y escalable para el sistema.

La adopción de una arquitectura sin servidor implica que el sistema no requiere la gestión directa de servidores físicos o virtuales. En su lugar, se utilizan servicios administrados en la nube, como AWS Lambda, para ejecutar el código de manera eficiente y a escala. AWS Lambda, por ejemplo, permite ejecutar fragmentos de código en respuesta a eventos, como solicitudes de los usuarios, sin necesidad de preocuparse por el aprovisionamiento y la administración de servidores. Esto proporciona una mayor agilidad y reduce la carga operativa para el equipo de desarrollo.

Además, la arquitectura sin servidor permite escalar automáticamente la infraestructura según la demanda. Esto significa que el sistema puede manejar picos de tráfico sin problemas, ya que los recursos se asignan y liberan de forma dinámica en función de la carga de trabajo. Por lo tanto, si hay un aumento repentino en el número de usuarios o solicitudes, la infraestructura se expande automáticamente para garantizar un rendimiento óptimo, y cuando la demanda disminuye, los recursos se reducen para optimizar los costos.

El uso de plataformas en la nube, como AWS, proporciona una infraestructura escalable y altamente disponible para soportar la arquitectura sin servidor. Estas plataformas cuentan con una red global de centros de datos, lo que garantiza una baja latencia y alta disponibilidad para los usuarios en diferentes ubicaciones geográficas. Además, ofrecen herramientas de monitoreo, escalado automático y gestión de recursos, lo que facilita el seguimiento y ajuste del rendimiento del sistema en tiempo real.

En resumen, se han alcanzado los objetivos planteados inicialmente mediante el uso de tecnologías y enfoques adecuados. La plataforma de computación en la nube y la arquitectura sin servidor han permitido la implementación eficiente y escalable del sistema. La aplicación web desarrollada cumple con su propósito de generar recetas de cocina a partir de las instrucciones del usuario, y se ha integrado una API de procesamiento del lenguaje natural para mejorar la generación de recetas. Estos logros demuestran el dominio de diferentes tecnologías y la capacidad para integrar conocimientos en la resolución de problemas, tanto a nivel profesional como personal.

8.2 Conocimientos adquiridos

A lo largo del desarrollo de mi proyecto, he trabajado con herramientas y tecnologías como AWS (Amazon Web Services), React, y otras mencionadas anteriormente. Estas herramientas eran desconocidas para mí o no había profundizado lo suficiente en ellas, pero debido a los requisitos y necesidades del proyecto, tuve que adquirir estos conocimientos de forma autodidacta.

Este proyecto ha sido una valiosa oportunidad para mi crecimiento personal, ya que me ha permitido desarrollar una aplicación web utilizando herramientas y tecnologías de vanguardia que son altamente relevantes en el mercado actual. Antes desconocía estos recursos, por lo que considero que ha sido una experiencia de gran importancia.

A continuación, destacaré algunos de los conocimientos adquiridos durante el desarrollo del proyecto, que considero importantes debido a su impacto en el proyecto o a nivel personal:

- Funcionamiento y utilización de herramientas de procesamiento de lenguaje natural.
- Conocimiento y análisis de diversos proveedores de servicios en la nube, incluyendo Amazon Web Services (AWS) y otros proveedores, lo que me permitió explorar a fondo sus características y diferencias.
- Profundización en el paradigma de las funciones sin servidor (serverless), mediante la práctica con servicios como AWS Lambda.
- Uso de frameworks front-end basados en JavaScript, como React, que me han brindado una nueva forma de acercarme al desarrollo web, con facilidades y desafíos específicos para abordar.
- Conocimiento y utilización de tecnologías de autenticación y autorización, como Amazon Cognito, y su integración en la aplicación desarrollada.
- Experiencia en el uso de herramientas para el análisis y monitoreo del rendimiento, como CloudWatch.
- Comunicación entre las distintas partes y servicios de una aplicación web.

- Utilización y familiarización con LaTeX, un sistema de composición de documentos que permite la creación de documentos técnicos.

En resumen, este proyecto ha sido muy beneficioso y enriquecedor para mí, ya que me ha permitido adquirir nuevos conocimientos y habilidades en las herramientas y tecnologías utilizadas. Ha sido una experiencia de aprendizaje significativa y ha contribuido a mi crecimiento personal y profesional.

8.3 Relación del trabajo desarrollado con los estudios cursados

- **Cloud computing**: La asignatura que más importancia ha tenido para este proyecto, durante el desarrollo de mi trabajo, pude aplicar los conocimientos adquiridos aquí para diseñar e implementar soluciones basadas en la nube. En esta asignatura comprendí los conceptos de Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS) etc. así como las arquitecturas escalables y los servicios serverless. Estos conocimientos fueron fundamentales para desplegar y gestionar eficientemente los recursos necesarios en mi proyecto.

- **Sistemas de almacenamiento y procesado distribuido**: Esta asignatura también fue de gran relevancia para el trabajo, ya que me proporcionó los fundamentos necesarios sobre manejar grandes volúmenes de datos y procesarlos de manera distribuida y como aplicar técnicas como el almacenamiento distribuido, la replicación y la partición de datos, lo cual resultó crucial para garantizar la escalabilidad y el rendimiento del sistema.

- **Machine Learning en entornos industriales y Sistemas inteligentes**: Gracias a los conocimientos adquiridos en estas asignaturas, a la hora de decidir qué tipo de intérprete de lenguaje natural utilizar para el proyecto, ha sido muy interesante contar con la base teórica obtenida sobre esta rama. Pudiendo distinguir entre sistemas expertos, sistemas basados en machine learning, árboles de decisión y demás herramientas, eligiendo aquella más útil y viable para el proyecto.

- **Ingeniería del Software**: La asignatura de Ingeniería del Software fue fundamental en mi trabajo, ya que me brindó las bases para desarrollar un proyecto sólido y de calidad. Aprendí sobre metodologías de desarrollo, buenas prácticas de programación, gestión de requisitos y pruebas de software. Estos conocimientos fueron aplicados en todas las etapas del desarrollo de mi proyecto, desde la planificación hasta la implementación y prueba del sistema.

- **Programación**: Las asignaturas de programación fueron fundamentales en el desarrollo del proyecto. A través de ellas, adquirí habilidades en diversos lenguajes de programación y paradigmas, lo que me permitió seleccionar la mejor opción para implementar las funcionalidades requeridas en mi proyecto. Además, aprendí a estructurar y organizar el código de manera eficiente, facilitando su mantenimiento y escalabilidad a lo largo del tiempo.

Además de estas asignaturas, que proporcionaron una base sólida de conocimientos técnicos y habilidades que fueron fundamentales para abordar los desafíos específicos del proyecto. Otras materias cursadas en el grado también han desempeñado un papel importante, como por ejemplo, desde asignaturas como Redes de Computadores y Seguridad en los sistemas informáticos que me permitieron comprender los principios y protocolos de comunicación en redes, así como las medidas de seguridad para proteger los datos y garantizar la integridad del sistemas. Hasta asignaturas como Gestión de Proyectos y Fundamentos de organización de empresas, que me brindaron habilidades adicionales en la planificación, organización y gestión de recursos, así como una comprensión de los aspectos económicos relacionados con el desarrollo de proyectos tecnológicos.

9 Trabajos futuros

Pese a que, tal y como se ha comentado en la sección 8, todos los objetivos del proyecto han sido alcanzados correctamente, siempre existe margen para la mejora y el rediseño. En este sentido, a pesar de los resultados obtenidos, existen áreas adicionales que podrían haberse explorado en futuros trabajos. Es importante tener en cuenta que este proyecto se ha desarrollado como Trabajo de Fin de Grado, con una carga lectiva de 12 créditos ECTS y plazos establecidos, lo cual ha limitado el alcance de los objetivos en función de una carga de trabajo máxima.

A continuación, se mencionan algunas de las posibles mejoras y ampliaciones que podrían haberse considerado en este trabajo:

Posibilidad de subir fotos y procesarlas

Una funcionalidad interesante que podría haberse incorporado es la capacidad de cargar imágenes y utilizar técnicas de procesamiento de imágenes y otras inteligencias artificiales para extraer información relevante. Por ejemplo:

- Dada una foto de una comida, el sistema podría identificar los ingredientes y proporcionar una posible receta basada en ellos.
- Del mismo modo, al recibir una foto de un ticket de compra, el sistema podría analizarlo y generar automáticamente una lista de ingredientes y recetas sugeridas.

Más funcionalidades para el usuario Se podría haber ampliado la interacción con el usuario mediante la adición de características como:

- La posibilidad de seleccionar varias de las recetas guardadas y generar automáticamente una lista de la compra. Esto facilitaría la planificación de las compras necesarias para preparar las recetas seleccionadas.
- Integración de redes sociales: Habilitar la opción de compartir recetas en redes sociales populares podría aumentar la visibilidad y la difusión de la aplicación. Los usuarios podrían compartir sus recetas favoritas, intercambiar ideas y recibir retroalimentación de otros amantes de la cocina.

- Convertidor de unidades: Para la adaptación de recetas a diferentes tamaños de porción.
- Menús: Generación de menús completos para eventos especiales.

Dominio web

Otro de los pasos importantes si queremos publicar la aplicación al mundo de forma oficial, es tener un dominio web.

Un dominio web es una dirección única en Internet que se utiliza para acceder a un sitio web. Por ejemplo, "www.ejemplo.com" es un dominio web. Los dominios web se utilizan para identificar y ubicar los recursos en línea, como sitios web, correos electrónicos y servicios en la nube.

Para obtener este dominio web, se podría utilizar Amazon Route 53, un servicio que ofrece registro de dominios y servicios de DNS (Sistema de Nombres de Dominio).

Estas son solo algunas ideas que podrían explorarse en futuros trabajos para mejorar y ampliar la funcionalidad del sistema. La continua evolución de las tecnologías y las necesidades de los usuarios abren infinitas posibilidades de expansión y enriquecimiento de la aplicación. Además, la arquitectura sin servidor utilizada en el desarrollo del sistema proporciona una ventaja adicional al permitir una mayor flexibilidad y escalabilidad. Gracias a esta arquitectura, agregar nuevas características y adaptarse a cambios futuros se vuelve más sencillo, ya que se pueden implementar funcionalidades de forma modular, sin preocuparse por la administración de la infraestructura subyacente. Esto permite que el sistema se ajuste de manera más eficiente a las necesidades cambiantes de los usuarios y garantiza una experiencia fluida y escalable.

Bibliografía

Cloudflare. (s.f.). *¿Qué es un generador de sitios estáticos?* [Cloudflare]. https://www.cloudflare.com/es-es/learning/performance/static-site-generator/. (Vid. pág. 16)

Deloitte. (2021). *¿Qué es Serverless Cloud?* https://www2.deloitte.com/es/es/blog/todo-tecnologia/2021/que-es-serverless-cloud.html. (Vid. pág. 5)

Devlin, J., & Chang, M.-W. (s.f.). *Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing* [Google AI Blog]. https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html. (Vid. pág. 17)

Flores, F. (s.f.). *Qué es Serverless, ventajas y servicios* [OpenWebinars]. https://openwebinars.net/blog/que-es-serverless-ventajas-y-servicios/. (Vid. pág. 1)

Geekflare. (s.f.). *8 Best AI Recipe Generators to Turn Ingredients into Cooked Food*. https://geekflare.com/ai-recipe-generators/. (Vid. pág. 11)

Handy, A. (s.f.). *Amazon introduces Lambda, Containers at AWS re:Invent* [SD Times]. https://sdtimes.com/amazon/amazon-introduces-lambda-containers/. (Vid. pág. 6)

IBM. (s.f.). *¿Qué es el procesamiento del lenguaje natural?* https://www.ibm.com/es-es/topics/natural-language-processing. (Vid. pág. 8)

Lance, M. C. (s.f.). Desarrollo de un sistema de reconocimiento automático del habla [RiuNet repositorio UPV], 27. https://riunet.upv.es/bitstream/handle/10251/10172/memoriaProyecto.pdf (vid. pág. 8)

Martínez, M. (s.f.). *Google BERT: actualización para entender el lenguaje natural* [Human Level]. https://www.humanlevel.com/articulos/posicionamiento-natural-buscadores/google-bert-actualizacion-para-entender-el-lenguaje-natural.html. (Vid. pág. 17)

Ríos, J. R. M., Tapia, J. A. H., Ordóñez, M. P. Z., & Segarra, M. J. C. (s.f.). Estado del arte: metodologías de desarrollo en aplicaciones web [3Ciencias]. https://www.3ciencias.com/articulos/articulo/estado-del-arte-metodologias-desarrollo-aplicaciones-web/ (vid. pág. 9)